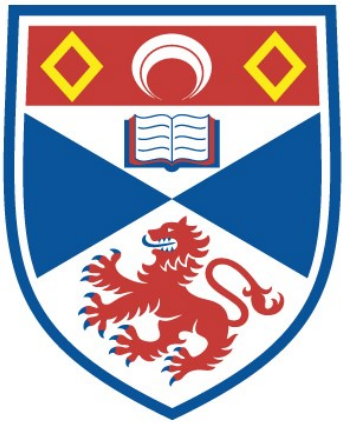
ALGORITHMS FOR SUBGROUP PRESENTATIONS: COMPUTER IMPLEMENTATION AND APPLICATIONS

Patricia M. Heggie

A Thesis Submitted for the Degree of PhD
at the
University of St Andrews



1991

ALGORITHMS FOR SUBGROUP PRESENTATIONS : COMPUTER IMPLEMENTATION AND APPLICATIONS

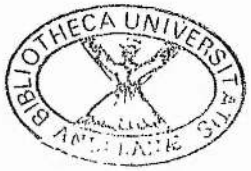
BY

PATRICIA M. HEGGIE

A thesis submitted for the degree of Doctor of Philosophy of the University of St. Andrews

Department of Mathematical and Computational Sciences

September 1990



ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

Abstract

One of the main algorithms of computational group theory is the Todd-Coxeter coset enumeration algorithm, which provides a systematic method for finding the index of a subgroup of a finitely presented group. This has been extended in various ways to provide not only the index of a subgroup, but also a presentation for the subgroup. These methods tie in with a technique introduced by Reidemeister in the 1920's and later improved by Schreier, now known as the Reidemeister-Schreier algorithm.

In this thesis we discuss some of these variants of the Todd-Coxeter algorithm and their inter-relation, and also look at existing computer implementations of these different techniques. We then go on to describe a new package for coset methods which incorporates various types of coset enumeration, including modified Todd-Coxeter methods and the Reidemeister-Schreier process. This also has the capability of carrying out Tietze transformation simplification. Statistics obtained from running the new package on a collection of test examples are given, and the various techniques compared.

Finally, we use these algorithms, both theoretically and as computer implementations, to investigate a particular class of finitely presented groups defined by the presentation :

$$\langle\ a, b\ \mid\ a^n = b^n = (ab^{-1})^\beta = 1,\ ab^2 = ba^2\ \rangle.$$

Some interesting results have been discovered about these groups for various values of β and n. For example, if n is odd, the groups turn out to be finite and metabelian, and if $\beta = 3$ or $\beta = 4$ the derived group has an order which is dependent on the values of n (mod 8) and n (mod 12) respectively.

Declarations

I Patricia M. Heggie hereby certify that this thesis has been composed by myself, that it is a record of my own work, and that it has not been accepted in partial or complete fulfilment of any other degree or professional qualification.

Signed Date 28/9/90

I was admitted to the Faculty of Science of the University of St. Andrews under Ordinance General No. 12 on 1/10/86 and as a candidate for the degree of Ph. D. on 1/10/87.

Signed . Date 28/9/90

In submitting this thesis to the University of St. Andrews I understand that I am giving permission for it to be made available for use in accordance with the regulations of the University Library for the time being in force, subject to any copyright vested in the work not being affected thereby. I also understand that the title and abstract will be published, and that a copy of the work may be made and supplied to any *bona fide* library or research worker.

Certificate

I hereby certify that the candidate has fulfilled the conditions of the Resolution and Regulations appropriate to the Degree of Ph. D.

Signature of Supervisor Date 28/8/1990

Acknowledgements

I would like to thank my supervisor, Dr. E.F.Robertson, for his encouragement and help over the past four years. I would also like to thank all of those people with infinite patience who put up with me and helped to keep me sane during the last few frantic months, especially Ian Gill and my mother !

I am also grateful to the Science and Engineering Research Council for their financial support over the first three years.

Introduction

Computational methods in group theory have been developed over a number of years, some designed specifically to help in the solution of one particular problem, and others to provide tools for solving general problems in group theory.

One of the main algorithms of computational group theory is the Todd-Coxeter coset enumeration algorithm, which provides a systematic method for finding the index of a subgroup of a finitely presented group. Although this was originally proposed as a method for hand calculation in 1936, it became one of the first algorithms to be implemented on electronic computers, and today is still one of the main tools of the computational group theorist.

The Todd-Coxeter algorithm has been extended in various ways to provide not only the index of a subgroup, but also a presentation for the subgroup. These methods tie in with a technique introduced by Reidemeister in the 1920's and later improved by Schreier, now known as the Reidemeister-Schreier algorithm.

In Chapter 1 of this thesis we discuss some of these variants of the Todd-Coxeter algorithm and their inter-relation. Then, in Chapter 2, we describe various existing computer implementations, and go on in Chapter 3 to describe a new package for coset methods which incorporates various types of coset enumeration, modified Todd-Coxeter methods, the Reidemeister-Schreier process and also Tietze transformation simplification techniques.

Chapter 4 contains statistics obtained from running the new package on a collection of test examples. In Chapter 5 we use both this computer implementation and hand calculations to investigate a particular class of finitely presented groups with the algorithms referred to above.

Contents

Chapter 1

Definitions and Group Algorithms

In this chapter we introduce some definitions and the theory of some group algorithms, the computer implementations of which will be discussed in the next chapter.

§1.1 Definitions

A group **G** is said to be generated by a subset $X=\{g_1, g_2,\ldots, g_n\}$ if each of its elements can be expressed as a product of members of the set $X^{\pm 1} = \{g_1, g_2, \ldots, g_n, g_1^{-1}, g_2^{-1}, \ldots, g_n^{-1}\}$. Such a product is called a word.

A freely reduced word in $g_1, g_2,\ldots, g_n$ is a word in which the symbols g_i^{ε}, $g_i^{-\varepsilon}$ ($\varepsilon = \pm 1$, $i = 1, 2,\ldots, n$) do not occur consecutively. A cyclically reduced word in $g_1, g_2,\ldots, g_n$ is a freely reduced word which does not begin with g_i^{ε} and end with $g_i^{-\varepsilon}$ ($\varepsilon = \pm 1$, $i = 1, 2,\ldots, n$).

Two words $w_1(g_i)$ and $w_2(g_i)$ are freely equal if they determine the same element of F_n, the free group on $g_1, g_2,\ldots, g_n$. We then write $w_1 \approx w_2$. Clearly, two words are freely equal if and only if one can be transformed into the other by insertions and deletions of the words $g_i g_i^{-1}$ or $g_i^{-1} g_i$ ($i = 1, 2,\ldots, n$).

If two words are equal the equation gives us a relation which holds in **G**. It is usual to write each relation in the form $R_j(g_1, g_2,\ldots, g_n) = 1$ where 1 denotes the identity element of **G**. Often the 1 is omitted and we work with relators instead of relations.

Suppose $R_1,\ldots,R_t$ are any relators of **G**. We say that the relator W is a consequence of $R_1,\ldots,R_t$ or derivable from $R_1,\ldots,R_t$ if we can apply the following operations a finite number of times and change W into the empty word, 1.

(i) Insertion of one of the words $R_1,\ldots,R_t, R_1^{-1},\ldots,R_t^{-1}$ or one of the trivial relators ($g_i g_i^{-1}$ or $g_i^{-1} g_i$, $i = 1, 2,\ldots, n$) between any two consecutive symbols of W, or before W or after W.

(ii) Deletion of any of the words $R_1,\ldots,R_t, R_1^{-1},\ldots,R_t^{-1}$ or a trivial relator if it forms a block of consecutive symbols in W.

Also we say that any two words $w_1(g_i)$ and $w_2(g_i)$ are equivalent if the operations (i) and (ii) applied a finite number of times change w_1 into w_2. In this case w_2 is derivable from w_1 and vice versa.

A set **R** of relations which hold in **G** defines the group **G** if every relation holding in **G** is a consequence of those in **R**.

In this case we say that **G** is presented by **X** and **R** and we write the presentation

$$\mathbf{G} = \langle \mathbf{X} \mid \mathbf{R} \rangle. \tag{1}$$

A group **G** is called finitely presented if it has a presentation of the form (1) such that both **X** and **R** are finite sets. It can be shown that every group has a presentation and every finite group can be finitely presented. [N.B. It is also possible for an infinite group to be finitely presented.]

A group presentation describes a group very concisely, but unlike a multiplication table, does not tell us the order of the group. This, however, can be found using a technique called coset enumeration (if the group is finite). Basically, coset enumeration counts the number of right cosets of a subgroup **H** of **G** in the group **G**. If the order of **H** is known, this gives us the order of **G** by Lagrange's Theorem. If **H** is taken to be the trivial subgroup, the order of **G** is immediate.

§1.2 Coset Enumeration

From the end of the last century, specific problems had been solved by coset enumeration, most of these involving much manipulative ingenuity. However, it wasn't until 1936 that the first general method for enumerating cosets was invented by J.A. Todd and H.S.M. Coxeter [34]. Their paper sets out a mechanical process which can be applied to any finite presentation. The original method was conceived for hand calculation only but later it was transferred with little modification to computers, thus greatly increasing the size of index and length of presentation that could be dealt with.

The description which follows is basically that given by Neubüser in [30] :

In the process, each coset is given a number. Coset 1 is always defined to be

the subgroup **H**. As the computation proceeds, each of the coset numbers $2, 3, 4, \ldots$ will be defined as a product αg_i^{ε} $(\varepsilon = \pm 1,\ 1 \le i \le n)$ where α is a previously defined coset number.

The whole method is based on two simple facts. If **H** is the group generated by the set $\{h_1, h_2, \ldots, h_s\}$ where each h_k, $1 \le k \le s$, is a word in the elements of $\mathbf{X}^{\pm 1}$, then $\mathbf{H}\, h_k = \mathbf{H}$, $1 \le k \le s$, and for any coset $\mathbf{H}\, g$, $g \in \mathbf{G}$, $\mathbf{H}\, g R_j(g_1, g_2, \ldots, g_n) = \mathbf{H}\, g$ $\forall\ R_j(g_1, g_2, \ldots, g_n) \in \mathbf{R}$, the set of relators.

We set up three sets of tables - one set for the subgroup generators, another for the group relations and a third called a coset table in which we keep a record of the definition of each coset number and any other equations of the form $\beta = \alpha g_i^{\varepsilon}$ which can be deduced from these.

Each "subgroup generator" table has only one row. The heading of the table is the subgroup generator written in expanded form in the generators, g_i (i.e. each element of the resulting word has exponent -1 or 1). If a heading has length c, then the table will have c+1 columns, each element of the heading occurring between two adjacent columns. The entry

	g_i	
α		β

indicates that the coset of **H** with number α in the Todd-Coxeter enumeration procedure multiplied by g_i from the right gives the coset with number β (and also that $\beta\, g_i^{-1} = \alpha$). Thus, the first and last entries in each subgroup table are 1 since $\mathbf{H}\, h_k = \mathbf{H}$, $1 \le k \le s$.

E.g. $h_1 = g_1 g_2 g_1^{-1} g_3$

g_1	g_2	g_1^{-1}	g_3
1			1

Similarly, for each relator $R_j(g_1, g_2, \ldots, g_n) \in \mathbf{R}$ a "relation table" is set up, in the same way as the subgroup generator tables but with one row for each coset number defined in the enumeration. Thus, at the beginning, we do not know in general how many rows will be needed. As each new coset number, λ, is defined, we add on a new row to each relation table, starting and finishing with this new coset number, λ.

	g_{i_1}	g_{i_2}	.....	g_{i_p}	
1					1
2					2
⋮					⋮
λ					λ

The third table has one column for each element in $X^{\pm 1}$ and these elements form the headings for the columns. The rows are indexed by each coset number which has been defined in the process. Into the "box" where row λ meets column g_i will be entered the coset number given to λg_i.

To begin the enumeration we choose an empty space immediately to the left or the right of some 1 in the subgroup generator or relation tables and fill it with the number 2. We set up row 2 of the coset table and record this definition $1g_i^{\varepsilon}=2$ and its consequence $2g_i^{-\varepsilon}=1$, then initialize row 2 in each of the relation tables. We go through each table and fill in the information $1g_i^{\varepsilon}=2$ and $2g_i^{-\varepsilon}=1$ in every possible place. When we can proceed no further we define coset number 3 in terms of coset 1 or 2 and carry on as before.

Whenever a row in the relation or subgroup generator tables closes, we get an equality of the form $\lambda g=\mu$ for some $g \in X^{\pm 1}$. This equality is called a <u>deduction</u>.

E.g.

.....	g_j	g_i	.....
	β		μ

We fill in $\beta g_j=\lambda$. This then gives us the new information $\lambda g_i=\mu$ and $\mu g_i^{-1}=\lambda$.

This gives rise to one of the following situations :

(i) The places of λg_i and μg_i^{-1} are both empty in the coset table. In this case we fill in the new information we have found and carry on as before, putting this information into all possible places in all the tables.

(ii) The place of λg_i in the coset table is already filled by μ (and hence μg_i^{-1} by λ). In this case our deduction gives us no new information, so we do nothing.

(iii) At least one of the places λg_i or μg_i^{-1} in the coset table is filled by a number different from that given by the deduction. This implies that our information is inconsistent and we have given two numbers to the same coset. This is called a coincidence. We then replace all occurrences of the larger number by the smaller in all of our tables. This may give rise to further coincidences which are dealt with in the same way.

The process has finished when all of the tables are complete. The order in which the coset definitions are made does not matter, although it may influence the efficiency of the enumeration (i.e. alter the number of coincidences found). In practice the smallest gaps in the subgroup generator and relation tables are usually filled in first. This gives us extra information in the form of deductions as quickly as possible, hopefully avoiding the definition of many unnecessary coset numbers.

N.B. In the above description, if there exists a generator of **G**, g_i say, such that g_i occurs in no group relation, then we must add the trivial relator $g_i g_i^{-1}$ (or $g_i^{-1} g_i$) to **R** to ensure that the columns headed by g_i and g_i^{-1} in the coset table are filled in.

We will now look at an easy example of the Todd Coxeter coset enumeration process which terminates without coincidences.

Example 1

$\mathbf{G} = \langle x, y \mid x^3 = y^3 = (xy)^2 = 1 \rangle$

$\mathbf{H} = \langle x \rangle$

The tables are set up initially as follows :

Subgroup

x
1 | 1

x	x	x
1 |		| 1

Relation Tables

y	y	y
1 |		| 1

x	y	x	y
1 |			| 1

Coset Table

	x	y	x^{-1}	y^{-1}
1				

1x=1 is a deduction gained straightaway from the subgroup table, so we can fill this in in all possible places. First of all we define 1y=2 and set up row 2 of each table (except the subgroup table which is finished). [In the working a dotted underline will denote a definition, a solid single underline a deduction where new information is obtained, and a double underline a coincidence.]

x
1 | 1

x	x	x
1	1	1
2		

y	y	y
1 | 2		1
2	1	2

x	y	x	y
1	1	2	1
2		1	2

	x	y	x^{-1}	y^{-1}
1	1	2	1	
2				1

Now we define 2x=3 in the first line of the third relation. This gives us the deduction 3y=1 which in turn gives us 2y=3 from the first row of relation 2.

x
1 | 1

x	x	x
1	1	1
2	3	
3		2

y	y	y
1 | 2 | 3		1
2	3	1
3	1	2

x	y	x	y
1	1	2 | 3	1
2	3	1	1
3		2	3

	x	y	x^{-1}	y^{-1}
1	1	2	1	3
2	3	3		1
3		1	2	2

Now define 3x=4 in the second row of the first relation table. This gives us the deduction 4x=2. When this is written into the last relation table we deduce that 4y=4.

x
1 | 1

x	x	x
1	1	1
2	3 | 4	2
3	4	2
4	2	3

y	y	y
1 | 2	3	1
2	3	1
3	1	2
4	4	4

x	y	x	y
1	1	2 | 3	1
2	3	1	1
3	4 | 4	2	3
4	2	3	4

	x	y	x^{-1}	y^{-1}
1	1	2	1	3
2	3	3	4	1
3	4	1	2	2
4	2	4	3	4

Thus all the tables are complete. Since we needed to define 4 coset numbers and there were no coincidences, $|\mathbf{G} : \mathbf{H}| = 4$. Since x has order 3 in **G**, $|\mathbf{H}| = 3$, so $|\mathbf{G}| = 12$.

Now an example with coincidences :

Example 2

$\mathbf{G} = \langle x, y \mid x^2y^3 = x^3y^4 = 1 \rangle$

$\mathbf{H} = \{1\}$

Here we are enumerating over the trivial subgroup, so there are no subgroup generator tables.

Relation Tables

x	x	y	y	y
1				1

x	x	x	y	y	y	y
1						1

Coset Table

	x	y	x^{-1}	y^{-1}
1				

First define $1x=2 \Rightarrow 2x^{-1}=1$ followed by $2x=3$ ($\Rightarrow 3x^{-1}=2$), $3y=4$ ($\Rightarrow 4y^{-1}=3$) and $4y=5$ ($\Rightarrow 5y^{-1}=4$). This gives us the deduction $5y=1$ from the first relation table (and its consequence $1y^{-1}=5$).

x	x	y	y	y	
1	2	3	4	5	1
2	3				2
3					3
4				3	4
5				4	5

x	x	x	y	y	y	y	
1	2	3		3	4	5	1
2	3						2
3							3
4						3	4
5					3	4	5

	x	y	x^{-1}	y^{-1}
1	2			5
2	3		1	
3		4	2	
4		5		3
5		1		4

Now define $3x=6$ giving the deduction $6y=3$ from the first row of the second relation. On filling this information into the second row of the first table, we gain the deduction $4y=2$. However, we defined $4y=5$ so we have a coincidence $5 \equiv 2$.

x	x	y	y	y	
1	2	3	4	5	1
2	3	6	3	4	2
3	6				3
4			6	3	4
5			3	4	5
6					6

x	x	x	y	y	y	y	
1	2	3	6	3	4	5	1
2	3	6					2
3	6						3
4						3	4
5					3	4	5
6							6

	x	y	x^{-1}	y^{-1}
1	2			5
2	3		1	
3	6	4	2	6
4		5		3
5		1		4
6		3	3	

Thus we replace all instances of 5 with 2 in the relation tables. In the coset table we can fill in $2y=1$ and $2y^{-1}=4$ since $5y=1$ and $5y^{-1}=4$.

x	x	y	y	y	
1	2	3	4	2	1
2	3	6	3	4	2
3	6			6	3
4			6	3	4
6					6

x	x	x	y	y	y	y	
1	2	3	6	3	4	2	1
2	3	6		6	3	4	2
3	6					6	3
4					6	3	4
6							6

	x	y	x^{-1}	y^{-1}
1	2			2
2	3	1	1	4
3	6	4	2	6
4		2		3
6		3	3	

[At this point we could have replaced all occurrences of 6 by 5 so that the coset numbers remain in sequence.]

Now define $6x=7$ giving the deduction $7y=6$ from the second row of the second relation. Putting this new information in the third row of the first table we get

the deduction $6y=6$. However, we already know that $6y=3$, so we have a coincidence $6 \equiv 3$. Also, we already have $7y=6$, so $7 \equiv 6$. Replacing all occurrences of 7 and 6 by 3 in the coset table we find both $2x=3$ and $3x=3$ implying that $2 \equiv 3$. Then, replacing all instances of 3 by 2 we find that $1x=2$ and $2x=2 \Rightarrow 1 \equiv 2$. Also, $2y^{-1}=4$ and $1y^{-1}=2$ so $4 \equiv 2$. Therefore, we have what is known as "total collapse", and we end up with one row in every table, each entry in that row being 1. This means that $\mathbf{G}=\mathbf{H}$, the trivial group, in this case.

We now prove that if **H** is of finite index, closure of all tables must be reached in a finite number of steps and the number of cosets left at the point of closure is the index of **G**. This is based on a proof by N.S. Mendelsohn [28].

First we prove :

Lemma 1

After a finite number of steps the first r rows of all the tables are stabilized i.e. none of the entries are further altered because of redundancy.

Proof

Use induction on the row number.

For r=1, it is obvious that the first row of each table eventually becomes stable since there are only a finite number of places and each is to be occupied by a positive integer. The effect of a redundancy (coincidence) is to replace some of these entries by smaller positive integers and this can happen only finitely often.

Suppose now that the first k rows are stabilized after a finite number of steps. Let i_k be the number of the next remaining line. Now there exists $m < i_k$ and $g \in \mathbf{X}^{\pm 1}$ such that $i_k = m\,g$ i.e. i_k occurs somewhere in the first k rows, so beyond this point no redundancy involves the replacement of i_k by a smaller integer. The argument used for the first row is now valid for the (k+1) st. ◊

Now suppose that the process continues indefinitely without closure and that at each coincidence, where coset number i disappears, we replace each instance of coset number $j > i$ in the tables by j-1. We then form a permutation representation of **G** on the set of all positive integers as follows. Let g_i be a generator and j any positive integer. Let R_v be one of the relations which starts with g_i and R_u one which ends with g_i. (Such relations can always be found by cyclically permuting or inverting any relation containing g_i.) After the jth row has become stable, let k be the second entry in the jth row in the table for R_v and let m be the second last entry in the jth row of the table for R_u. Then $j\,g_i = k$ and $m\,g_i = j$. Now associate the permutation P_{g_i} with g_i, where $P_{g_i} : j \rightarrow k$ and $m \rightarrow j$. Since j is an arbitrary positive number and appears both as an image and a pre-image, P_{g_i} is a permutation of all the positive integers. Let P_G be the group generated by all the P_{g_i}. Then the mapping $g_i \rightarrow P_{g_i}$ is a homomorphism of **G** onto P_G.

We now show that P_G is a transitive group. In fact, we show that every integer is in the orbit of 1.

If this was not true, let M be an orbit and u its smallest member. If $u \neq 1$ then its first appearance in the tables is to the left or right of an integer $v < u$. This means that there is a g_i (or g_i^{-1}) in **G** such that P_{g_i} (or $P_{g_i}^{-1}$) : $v \rightarrow u$. Hence v is in M – a contradiction.

Now from the subgroup generator tables we see that for every generator of **H** the corresponding element of P_G fixes 1. It is clear that no element outside **H** fixes 1.

Thus, we can interpret P_G as the representation of **G** by the permutations induced on the cosets of **H** by right multiplication by elements of **G**, but, since P_G is transitive on infinitely many elements, the order of P_G is infinite, and hence the index of **H** is infinite, a contradiction.

Thus, closure of all tables occurs after finitely many steps.

In the process, the numbers represent cosets of **H**. Each of these has been multiplied by each generator of **G** and its inverse and the cosets obtained from these multiplications have been assigned numbers. Hence, each element of **G** lies in one of these numbered cosets. Therefore, if the process terminates with a set K of k coset numbers, we know that $|\mathbf{G} : \mathbf{H}| \leq k$.

Since, from the coset table, $\alpha g_i = \beta$ iff $\beta g_i^{-1} = \alpha$, we see that for each of the k coset numbers an image under each of $g_1, \ldots, g_n, g_1^{-1}, \ldots, g_n^{-1}$ has been defined i.e. to each generator g_i a permutation $\overline{g_i}$ of the set K has been assigned. Further, $\overline{g_1}, \ldots, \overline{g_n}$ generate a transitive group on K. From the relation tables we see that these $\overline{g_i}$ satisfy the relations of **G**, hence **G** acts transitively on K via the homomorphism $g_i \rightarrow \overline{g_i}$ so $|\mathbf{G} : \mathrm{Stab}_G(1)| = k$. The subgroup tables show that $\mathbf{H} \leq \mathrm{Stab}_G(1)$ giving $|\mathbf{G} : \mathbf{H}| \geq k$, so finally $|\mathbf{G} : \mathbf{H}| = k$.

Thus, when a Todd-Coxeter process terminates, it determines the index $|\mathbf{G} : \mathbf{H}|$ and a permutation representation on the right cosets of **H**.

However, this result does not help us much in practice. We know that if $|\mathbf{G} : \mathbf{H}|$ is finite, the Todd-Coxeter process will terminate after a finite number of steps, but no indication is given of how many cosets will need to be defined in order to reach closure. If we could give a bound for the number of cosets necessary (e.g. as a function based on the number and lengths of the relators, the subgroup generators and the size of the index) this would turn the Todd-Coxeter process into a proper algorithm. The fact that such an algorithm does not exist has been proved [31].

In some enumerations it is not enough just to define cosets in the relation and subgroup tables. Examples can be constructed [36] such that this leaves "holes" in the early rows of the coset table which are never filled. Thus, the first k rows of the coset table do not close so, since Lemma 1 is not satisfied, a finite index is never found. This problem can be solved by defining new cosets in the coset table to fill these holes when it is noticed that this situation is developing. The original Todd-Coxeter paper got round this problem by insisting that each coset number defined is written into each of the relation tables in every possible column before the next coset is defined.

§1.3 Tietze Transformations

A group **G** can have many presentations, for, given a set of generators for **G**, there are many possible sets of defining relators.

In 1908 H.Tietze showed that given a presentation

$$G = \langle g_1, g_2, \ldots, g_n \mid R_1 = R_2 = \ldots = R_t = 1 \rangle \qquad (2)$$

for a group **G**, any other presentation for **G** can be obtained by repeated applications of certain transformations (T1), (T2), (T3) and (T4) to (2). The following description of these is based on the descriptions in [23] and [27].

Let $\overline{\mathbf{R}}$ be the normal closure of **R** in **F**, the free group on the elements of **X**. In each case we transform the presentation $\langle \mathbf{X} \mid \mathbf{R} \rangle$ to the new one $\langle \mathbf{X}' \mid \mathbf{R}' \rangle$ of the same group.

(T1) Adjoining a relator

Here we add the relator r, $r \in \overline{\mathbf{R}} \backslash \mathbf{R}$. Then $\mathbf{X}' = \mathbf{X}$ and $\mathbf{R}' = \mathbf{R} \cup \{r\}$.

(T2) Removing a relator

When a relator r is derivable from the others, i.e. $r \in \mathbf{R} \cap \overline{\mathbf{R} \backslash \{r\}}$ we can obviously remove it without changing the group. So $\mathbf{X}' = \mathbf{X}$ and $\mathbf{R}' = \mathbf{R} \backslash \{r\}$.

(T3) Adjoining a generator

If w is any word in the generators $g_1, g_2, \ldots, g_n$ of **G** we can introduce a new generating symbol z, say, and adjoin the relation $z^{-1}w = 1$ to **R**. Thus, $\mathbf{X}' = \mathbf{X} \cup \{z\}$ and $\mathbf{R}' = \mathbf{R} \cup \{z^{-1}w\}$.

(T4) Removing a generator

If one of the defining relations in (2) takes the form $z = w$, $z \in \mathbf{X}$, where w is a word in the elements of $\mathbf{X}^{\pm 1}$ other than z, and this is the only relation that z occurs in, we can delete z from the set of generators and remove the relator $z^{-1}w$ from **R**. So, $\mathbf{X}' = \mathbf{X} \backslash \{z\}$ and $\mathbf{R}' = \mathbf{R} \backslash \{z^{-1}w\}$.

These four transformations (T1), (T2), (T3) and (T4) are called Tietze transformations. It is clear that (T1), (T2) and (T3) do not change the group, so, since (T4) is the inverse of (T3), it must also conserve the group.

$$\langle \mathbf{X} \mid \mathbf{R} \rangle \overset{(T3)}{\rightarrow} \langle \mathbf{X}, z \mid \mathbf{R}, z^{-1}w \rangle \overset{(T4)}{\rightarrow} \langle \mathbf{X} \mid \mathbf{R} \rangle$$

It is possible to replace (T4) by a slightly more useful definition :

(T4)' Removing a generator

If one of the defining relations takes the form $z = w$, $z \in \mathbf{X}$, where w is a word in the elements of $\mathbf{X}^{\pm 1}$ other than z we can delete z from the set of generators, remove the relator $z^{-1}w$ from the set of relators and replace z by w in the remaining defining relators.

Theorem 2

Given two finite presentations of the same group, one can be obtained from the other by a finite sequence of Tietze transformations.

Proof

Given two such presentations

$$\mathbf{G} \cong \langle \mathbf{X} \mid \mathbf{R}(\mathbf{X}) = 1 \rangle \cong \langle \mathbf{Y} \mid \mathbf{S}(\mathbf{Y}) = 1 \rangle$$

suppose that $\mathbf{X} = \mathbf{X}(\mathbf{Y})$, $\mathbf{Y} = \mathbf{Y}(\mathbf{X})$ are two systems of equations expressing the generators $\mathbf{X}$ in terms of the generators $\mathbf{Y}$ and vice versa. We now apply Tietze transformations en bloc to the first presentation.

	Generators	Relations
	X	R(X)=1
(1) (T3)	X, Y	R(X)=1, Y=Y(X)
(2) (T1)	X, Y	R(X)=1, Y=Y(X), X=X(Y)
(3) (T1)	X, Y	R(X)=1, Y=Y(X), X=X(Y), R(X(Y))=1
(4) (T2)	X, Y	Y=Y(X), X=X(Y), R(X(Y))=1
(5) (T1)	X, Y	Y=Y(X), X=X(Y), R(X(Y))=1, Y=Y(X(Y))
(6) (T2)	X, Y	X=X(Y), R(X(Y))=1, Y=Y(X(Y))
(7) (T4)	Y	R(X(Y))=1, Y=Y(X(Y))
(8) (T1)	Y	R(X(Y))=1, Y=Y(X(Y)), S(Y)=1
(9) (T2)	Y	S(Y)=1

Notes

Let $\mathbf{X}=\{x_1, x_2,\ldots, x_n\}$, $\mathbf{Y}=\{y_1, y_2,\ldots, y_s\}$.

(5) $\rightarrow$ (6) Here the first set of relations can be removed since it contains the same information as $\mathbf{X}=\mathbf{X}(\mathbf{Y})$ and $\mathbf{Y}=\mathbf{Y}(\mathbf{X}(\mathbf{Y}))$ together.

(6) $\rightarrow$ (7) In line (6), the first set of relations is the only one left containing any elements of $\mathbf{X}$. It consists of relations of the form $x_i = y_{i_1}y_{i_2}\ldots y_{i_k}$. Therefore, the generators x_i can be removed by (T4).

(8) $\rightarrow$ (9) Here $\{\, \mathbf{R}(\mathbf{X}(\mathbf{Y}))=1,\ \mathbf{Y}=\mathbf{Y}(\mathbf{X}(\mathbf{Y}))\,\}$ is a set of defining relations for $\mathbf{G}$ in terms of the generating set $\mathbf{Y}$. We then add $|\mathbf{S}|$ new relations in the elements of $\mathbf{Y}$ which also define $\mathbf{G}$. Thus, $\mathbf{R}(\mathbf{X}(\mathbf{Y}))=1$ and $\mathbf{Y}=\mathbf{Y}(\mathbf{X}(\mathbf{Y}))$ must be derivable from $\mathbf{S}(\mathbf{Y})=1$ so can be removed. Thus, $\langle\, \mathbf{X} \mid \mathbf{R}(\mathbf{X})=1 \,\rangle$ can be transformed into $\langle\, \mathbf{Y} \mid \mathbf{S}(\mathbf{Y})=1 \,\rangle$ under $4|\mathbf{Y}| + 3|\mathbf{R}| + 2|\mathbf{X}| + |\mathbf{S}|$ Tietze transformations. ◊

Given a presentation for $\mathbf{G}$ in terms of $\mathbf{X}$ and asked to represent it on a new generating set $\mathbf{Y}$, line (7) gives us the new presentation :

$$\mathbf{G} = \langle\, \mathbf{Y} \mid \mathbf{R}(\mathbf{X}(\mathbf{Y}))=1\ ,\ \mathbf{Y}=\mathbf{Y}(\mathbf{X}(\mathbf{Y})) \,\rangle \qquad (3)$$

not $\mathbf{G}=\langle\, \mathbf{Y} \mid \mathbf{R}(\mathbf{X}(\mathbf{Y}))=1 \,\rangle$ as one might expect !

There is no general algorithm for deciding whether two given finite presentations define isomorphic groups. To transform one presentation into the other, one must use a certain amount of intuition and hope for some luck !

§1.4 Presentations of Subgroups

Let $\mathbf{G}$ be a group with the presentation

$$\mathbf{G} = \langle\, g_1, g_2,\ldots, g_n \mid R_1(g_1, g_2,\ldots, g_n) = \ldots = R_t(g_1, g_2,\ldots, g_n) = 1 \,\rangle. \qquad (4)$$

To find a presentation for a subgroup $\mathbf{H}$ of $\mathbf{G}$ we need to find words in the g_i which generate $\mathbf{H}$ and we also need to find a process for "rewriting" a word in the g_i, defining an element of $\mathbf{H}$, as a word in the generators of $\mathbf{H}$.

Let $\mathbf{G}$ be presented as in (4) and let $\mathbf{H}$ be the subgroup of $\mathbf{G}$ generated by the

words $J_k(g_i)$. Then a rewriting process for **H**, (w.r.t. the generators $J_k(g_i)$), is a mapping

$$\tau : U(g_i) \rightarrow V(h_k) \tag{5}$$

of words $U(g_i)$ defining elements of **H** into the symbols h_k such that the words $U(g_i)$ and $V(h_k)$ define the same element of **H**. [The symbols h_k will be the generating symbols used for $J_k(g_i)$ in our resulting presentation for **H**.]

In this work I will insist that a rewriting process also satisfies the following two conditions :

(1) $\tau(U) \approx \tau(U^*)$ if $U(g_i)$ and $U^*(g_i)$ are freely equal words which define elements of **H**.

(2) $\tau(U_1.U_2) \approx \tau(U_1)\tau(U_2)$ where $U_1(g_i)$ and $U_2(g_i)$ define elements of **H**.

This definition of a rewriting process is stronger than that defined in [27, p.86] and saves some work in the proof of the theorems which follow.

Theorem 3

Let **H** be a subgroup of **G** presented as in (4). If $J_k(g_i)$ are generators for **H** and the mapping τ is a rewriting process for **H** (w.r.t the generators $J_k(g_i)$) then a presentation for **H** under the mapping $h_k \rightarrow J_k(g_i)$ is obtained by using the symbols h_k as generating symbols and using the following equations as defining relations :

$$h_k = \tau(J_k(g_i)) \tag{6}$$

$$\tau(w R_j w^{-1}) = 1 \tag{7}$$

where $R_j(g_i)$ is a defining relator in (4) and w is any word in the group generators, g_i.

Proof

We first show that (6) and (7) are relations in the subgroup, **H**. $\tau(J_k(g_i))$ defines the same element of **H** as $J_k(g_i)$ by the definition of a rewriting process. Therefore, since h_k is just the defining symbol for $J_k(g_i)$, the relations (6) clearly hold in **H**. In condition (2) if we replace both U_1 and U_2 by the empty word 1 we obtain $\tau(1) = \tau(1).\tau(1)$ so $\tau(1) = 1$. Now, looking at relations (7), $w R_j w^{-1} = 1 \ \forall \ w \in G$ and $R_j \in R$, so $\tau(wR_jw^{-1}) = \tau(1) = 1$. Thus, the relations (7) also hold in **H**.

Now to show that (6) and (7) are defining relations, we must show that any relator in **H**,

$$h_{k_1}^{\varepsilon_1} \dots h_{k_r}^{\varepsilon_r} \quad (\varepsilon_i = \pm 1) \tag{8}$$

can be reduced to the empty word using equations (6) and (7) and conditions (1) and (2).

In condition (2) replace U_2 by U_1^{-1} to get

$$\tau(1) = \tau(U_1)\,\tau(U_1^{-1}), \text{ or equivalently}$$

$$\tau(U_1^{-1}) = \tau(U_1)^{-1}. \tag{9}$$

Then, using (9) and condition (2), we derive

$$\tau(U_1^{\varepsilon_1} \dots U_p^{\varepsilon_p}) = \tau(U_1)^{\varepsilon_1} \dots \tau(U_p)^{\varepsilon_p}. \tag{10}$$

Now, using (6), (8) may be replaced by

$$\tau(J_{k_1}(g_i))^{\varepsilon_1} \dots \tau(J_{k_r}(g_i))^{\varepsilon_r} \tag{11}$$

which by (10) can be further replaced by

$$\tau(J_{k_1}^{\varepsilon_1} \dots J_{k_r}^{\varepsilon_r}). \tag{12}$$

Now, since (8) is a relator under $h_k \to J_k(g_i)$,

$$J_{k_1}(g_i)^{\varepsilon_1} \dots J_{k_r}(g_i)^{\varepsilon_r} \tag{13}$$

is equal to the identity of **H** and hence that of **G**. Since **G** has presentation (4) it can be seen that (13) is freely equal to a product

$$(w_1 R_{j_1} w_1^{-1})^{\eta_1} \dots (w_v R_{j_v} w_v^{-1})^{\eta_v} \in \overline{\mathbf{R}}$$

where $\eta_i = \pm 1$ and $R_{j_i} \in \mathbf{R}$ and w_i is a word in the elements of $\mathbf{X}^{\pm 1}$. Now, using condition (1), (12) can be replaced by

$$\tau(\,(w_1 R_{j_1} w_1^{-1})^{\eta_1} \dots (w_v R_{j_v} w_v^{-1})^{\eta_v}\,)$$

which can be further replaced by $\tau(w_1 R_{j_1} w_1^{-1})^{\eta_1} \dots \tau(w_v R_{j_v} w_v^{-1})^{\eta_v}$ using equation (10). But then the relations in (7) allow us to reduce this to the empty word. Thus, the expression in (12), and hence that in (8), is equivalent to the empty word. ◊

The presentation obtained here is very cumbersome. By a careful choice of generators and rewriting process, the presentation can be greatly simplified. In the

rest of this chapter we are concerned with using a "right coset function" **G** mod **H** to find both the subgroup generators and a suitable rewriting process.

If **G** has the presentation (4), then a right coset function for **G** on the generators g_i modulo a subgroup **H**, is a mapping of the words in g_i

$$w(g_i) \rightarrow \overline{w}(g_i)$$

where the $\overline{w}(g_i)$ form a right coset representative system for **G** mod **H**, containing the empty word as the coset representative of **H** itself, and where $\overline{w}(g_i)$ is the coset representative for the coset containing $w(g_i)$. [Note that I am using a bar ($\overline{\ }$) to denote a coset representative and also the normal closure of a set. Which of the two is intended should be clear from the context.]

Before stating the main theorem, I will prove a useful result about a right representative function :

By definition w and $\overline{w}$ lie in the same coset of **H**. Thus, w v and $\overline{w}$ v also lie in the same coset, so by the definition of a coset representative

$$\overline{w\,v} = \overline{\overline{w}\,v}. \tag{14}$$

Theorem 4

If $w \rightarrow \overline{w}$ is a right coset function for **G** mod **H**, then **H** is generated by the words

$$K\,g_i\,\overline{K\,g_i}^{-1} \tag{15}$$

where K is an arbitrary representative and g_i an arbitrary generator of **G**.

Proof

$K\,g_i$ and $\overline{K\,g_i}$ determine the same right coset of **H**, therefore $K\,g_i\,\overline{K\,g_i}^{-1} \in \mathbf{H}$. Since K is a representative, it follows by (14) that $\overline{\overline{K\,g_i^{-1}}\,g_i} = \overline{K\,g_i^{-1}\,g_i} = \overline{K} = K$. Now, setting $M = \overline{K\,g_i^{-1}}$,

$$M\,g_i\,\overline{M\,g_i}^{-1} = \overline{K\,g_i^{-1}}\,g_i\,K^{-1} = (K\,g_i^{-1}\,\overline{K\,g_i^{-1}}^{-1})^{-1}.$$

Hence, $K\,g_i^{-1}\,\overline{K\,g_i^{-1}}^{-1}$ is the inverse of the word $M\,g_i\,\overline{M\,g_i}^{-1}$ which is included in (15).

Suppose

$$U = g_{i_1}^{\varepsilon_1} \dots g_{i_r}^{\varepsilon_r} \tag{16}$$

defines an element of **H**. We must show that we can express U in terms of the words in (15). We insert before and after each $g_{i_j}^{\varepsilon_j}$ the words $\overline{w}_j$ and $\overline{w_j g_{i_j}^{\varepsilon_j}}^{-1}$ respectively and try to choose the w_j so that our new product,

$$\overline{w}_1 g_{i_1}^{\varepsilon_1} \overline{w_1 g_{i_1}^{\varepsilon_1}}^{-1} \cdot \overline{w}_2 g_{i_2}^{\varepsilon_2} \overline{w_2 g_{i_2}^{\varepsilon_2}}^{-1} \cdot \dots\dots \cdot \overline{w}_r g_{i_r}^{\varepsilon_r} \overline{w_r g_{i_r}^{\varepsilon_r}}^{-1} \tag{17}$$

defines the same element of **H** as (16).

If we choose

$$w_1 = 1, \; w_2 = w_1 g_{i_1}^{\varepsilon_1}, \; w_3 = w_2 g_{i_2}^{\varepsilon_2}, \dots\dots, \; w_r = w_{r-1} g_{i_{r-1}}^{\varepsilon_{r-1}}$$

this clearly holds,

i.e. $\quad w_1 = 1, \; w_2 = g_{i_1}^{\varepsilon_1}, \; w_3 = g_{i_1}^{\varepsilon_1} g_{i_2}^{\varepsilon_2}, \dots\dots, \; w_r = g_{i_1}^{\varepsilon_1} g_{i_2}^{\varepsilon_2} \dots g_{i_{r-1}}^{\varepsilon_{r-1}}$.

Then, (17) is freely equal to

$$\overline{w}_1 \; U \; \overline{w_r g_{i_r}^{\varepsilon_r}}^{-1} = \overline{1} \, U \, \overline{U}^{-1} = 1 \, U \, 1^{-1} = U \, .$$

It is clear that each $\overline{w}_j g_{i_j}^{\varepsilon_j} \overline{w_j g_{i_j}^{\varepsilon_j}}^{-1}$ is one of the words in (15) or its inverse (which we have shown to also be one of the words in (15)). Thus, the words in (15) generate **H**. ◊

Corollary

Let $w \to \overline{w}$ be a right coset representative function for **G** mod **H**. Introduce the generating symbol s_{K,g_i} for the element of **H** defined by the word $K g_i \overline{K g_i}^{-1}$ where K is an arbitrary right coset representative and g_i is a generator in (4). Define the mapping τ of the word U in (16), which defines an element of **H**, by

$$\tau(U) = s_{K_1, g_{i_1}}^{\varepsilon_1} s_{K_2, g_{i_2}}^{\varepsilon_2} \dots\dots s_{K_r, g_{i_r}}^{\varepsilon_r} \tag{18}$$

where K_j is the representative of the initial segment of U preceding g_{i_j} if $\varepsilon_j = 1$ and K_j is the representative of the initial segment of U up to and including $g_{i_j}^{-1}$ if $\varepsilon_j = -1$. Then τ is a rewriting process for **H**.

Proof

First of all we show that if $s_{K_j, g_{i_j}}$ is replaced by $K_j g_{i_j} \overline{K_j g_{i_j}}^{-1}$ in (18) then the resulting word in terms of the generators g_i defines the same element of **H** as $U(g_i)$.

If w_j is the initial segment of U preceding $g_{i_j}^{\varepsilon_j}$ then from the definition of K_j, and using (14) :

$$\text{If } \varepsilon_j = 1,\ K_j = \overline{w_j} \Rightarrow s_{K_j, g_{i_j}} = \overline{w_j} g_{i_j} \overline{\overline{w_j} g_{i_j}}^{-1} = \overline{w_j} g_{i_j} \overline{w_j g_{i_j}}^{-1}$$

and if $\varepsilon_j = -1,\ K_j = \overline{w_j g_{i_j}^{-1}} \Rightarrow s_{K_j, g_{i_j}} = \overline{w_j g_{i_j}^{-1}}\, g_{i_j} \overline{\overline{w_j g_{i_j}^{-1}} g_{i_j}}^{-1} = \overline{w_j g_{i_j}^{-1}}\, g_{i_j} \overline{w_j}^{-1}$

$$\Rightarrow s_{K_j, g_{i_j}}^{-1} = \overline{w_j} g_{i_j}^{-1} \overline{w_j g_{i_j}^{-1}}^{-1}$$

Hence, in both cases, $s_{K_j, g_{i_j}}^{\varepsilon_j}$ is replaced by $\overline{w_j} g_{i_j}^{\varepsilon_j} \overline{w_j g_{i_j}^{\varepsilon_j}}^{-1}$ so (18) becomes (17) which defines the same element of **H** as $U(g_i)$.

Now we have to show that conditions (1) and (2) hold for τ.

(1) If U and U^* are freely equal but not identical words there occurs at least one generator, g_{i_j} say, in U which is adjacent to its inverse.

Let $U = w_1 g_{i_j} g_{i_{j+1}}^{-1} w_2$ and $U^* = w_1 w_2$ where $g_{i_j} = g_{i_{j+1}}$, $w_1 = g_{i_1} \cdots g_{i_{j-1}}$ and $w_2 = g_{i_{j+2}} \cdots g_{i_p}$. Then,

$$\tau(U) = s_{K_1, g_{i_1}}^{\varepsilon_1} s_{K_2, g_{i_2}}^{\varepsilon_2} \cdots\cdots s_{K_{j-1}, g_{i_{j-1}}}^{\varepsilon_{j-1}} \left[s_{K_j, g_{i_j}} s_{K_{j+1}, g_{i_{j+1}}}^{-1} \right] s_{K_{j+2}, g_{i_{j+2}}}^{\varepsilon_{j+2}} \cdots\cdots s_{K_p, g_{i_p}}^{\varepsilon_p}$$

and

$$\tau(U^*) = s_{K_1, g_{i_1}}^{\varepsilon_1} s_{K_2, g_{i_2}}^{\varepsilon_2} \cdots\cdots s_{K_{j-1}, g_{i_{j-1}}}^{\varepsilon_{j-1}} s_{K_{j+2}, g_{i_{j+2}}}^{\varepsilon_{j+2}} \cdots\cdots s_{K_p, g_{i_p}}^{\varepsilon_p} .$$

Now since $\varepsilon_j = 1$, $K_j = \overline{w_1}$ and since $\varepsilon_{j+1} = -1$, $K_{j+1} = \overline{w_1 g_{i_j} g_{i_{j+1}}^{-1}} = \overline{w_1}$ since $g_{i_j} = g_{i_{j+1}}$.

Thus, $s_{K_{j+1}, g_{i_{j+1}}}^{-1}$ can be replaced by

$$\left[K_{j+1} g_{i_{j+1}} \overline{K_{j+1} g_{i_{j+1}}}^{-1} \right]^{-1} = \left[\overline{w_1}\, g_{i_{j+1}} \overline{w_1\, g_{i_{j+1}}}^{-1} \right]^{-1} = \overline{w_1\, g_{i_j}} g_{i_j}^{-1} \overline{w_1}^{-1} .$$

Hence, the bracketed section in $\tau(U)$ can be replaced by

$$\overline{w_1} g_{i_j} \overline{w_1 g_{i_j}}^{-1} \cdot \overline{w_1 g_{i_j}} g_{i_j}^{-1} \overline{w_1}^{-1} \approx 1.$$

Likewise, if $U = w_1 g_{i_j}^{-1} g_{i_{j+1}} w_2$ where $g_{i_j} = g_{i_{j+1}}$ then

$$\tau(U) = s_{K_1, g_{i_1}}^{\varepsilon_1} s_{K_2, g_{i_2}}^{\varepsilon_2} \cdots\cdots s_{K_{j-1}, g_{i_{j-1}}}^{\varepsilon_{j-1}} \left[s_{K_j, g_{i_j}}^{-1} s_{K_{j+1}, g_{i_{j+1}}} \right] s_{K_{j+2}, g_{i_{j+2}}}^{\varepsilon_{j+2}} \cdots\cdots s_{K_p, g_{i_p}}^{\varepsilon_p}.$$

This time $\varepsilon_j = -1$ so $K_j = \overline{w_1 g_{i_j}^{-1}}$, and $\varepsilon_{j+1} = 1$ so $K_{j+1} = \overline{w_1 g_{i_j}^{-1}}$.

Therefore, $s_{K_j, g_{i_j}}^{-1} s_{K_{j+1}, g_{i_{j+1}}}$ can be replaced by

$$\left[K_j g_{i_j} \overline{K_j g_{i_j}}^{-1} \right]^{-1} K_{j+1} g_{i_{j+1}} \overline{K_{j+1} g_{i_{j+1}}}^{-1}$$

$$= \left[\overline{w_1 g_{i_j}^{-1}} \; g_{i_j} \; \overline{w_1 g_{i_j}^{-1} g_{i_j}}^{-1} \right]^{-1} \overline{w_1 g_{i_j}^{-1}} \; g_{i_{j+1}} \; \overline{w_1 g_{i_j}^{-1} g_{i_{j+1}}}^{-1} \quad \text{using (14)}$$

$$= \overline{w_1} \; g_{i_j}^{-1} \; \overline{w_1 g_{i_j}^{-1}}^{-1} \; \overline{w_1 g_{i_j}^{-1}} \; g_{i_j} \; \overline{w_1}^{-1} \quad \text{since } g_{i_j} = g_{i_{j+1}}$$

$$\approx 1.$$

K_{j+2} is equal to $\overline{w_1 g_{i_j} g_{i_j}^{-1}} = \overline{w_1}$ in the first case, or $\overline{w_1 g_{i_j}^{-1} g_{i_j}} = \overline{w_1}$ in the second, and so $\tau(U) \approx \tau(U^*)$ in each case.

---

(2) Let $U_1 = g_{i_1}^{\varepsilon_{i_1}} g_{i_2}^{\varepsilon_{i_2}} \cdots g_{i_r}^{\varepsilon_{i_r}}$, $U_2 = g_{j_1}^{\varepsilon_{j_1}} g_{j_2}^{\varepsilon_{j_2}} \cdots g_{j_p}^{\varepsilon_{j_{p-1}}}$.

Now, $\tau(U_1 U_2) = s_{K_{l_1}, g_{l_1}}^{\varepsilon_{l_1}} \cdots\cdots s_{K_{l_r}, g_{l_r}}^{\varepsilon_{l_r}} s_{K_{l_{r+1}}, g_{l_{r+1}}}^{\varepsilon_{l_{r+1}}} \cdots\cdots s_{K_{l_v}, g_{l_v}}^{\varepsilon_{l_v}}$

where $U_1 U_2$ is not freely reduced and $l_n = i_n$, $1 \le n \le r$, $l_n = j_{n-r}$, $r+1 \le n \le v$.

Clearly, $\tau(U_1) = s_{K_{l_1}, g_{l_1}}^{\varepsilon_{l_1}} \cdots\cdots s_{K_{l_r}, g_{l_r}}^{\varepsilon_{l_r}}$ since the values of $s_{K_{l_n}, g_{l_n}}^{\varepsilon_{l_n}}$ do not depend on the segment of the word $U_1 U_2$ following $g_{l_n}^{\varepsilon_{l_n}}$.

$$s_{K_{l_{r+1}}, g_{l_{r+1}}}^{\varepsilon_{l_{r+1}}} = \left[K_{l_{r+1}} g_{l_{r+1}} \overline{K_{l_{r+1}} g_{l_{r+1}}}^{-1} \right]^{\varepsilon_{l_{r+1}}}$$

$$= \overline{g_{i_1}^{\varepsilon_{i_1}} g_{i_2}^{\varepsilon_{i_2}} \cdots g_{i_r}^{\varepsilon_{i_r}} \; g_{j_1}^{\varepsilon_{j_1}}} \; \overline{g_{i_1}^{\varepsilon_{i_1}} g_{i_2}^{\varepsilon_{i_2}} \cdots g_{i_r}^{\varepsilon_{i_r}} g_{j_1}^{\varepsilon_{j_1}}}^{-1}$$

$$= \overline{U_1 g_{j_1}^{\varepsilon_{j_1}}} \; \overline{U_1 g_{j_1}^{\varepsilon_{j_1}}}^{-1}$$

$$= g_{j_1}^{\varepsilon_{j_1}} \overline{g_{j_1}^{\varepsilon_{j_1}}}^{-1} \qquad \text{since by (14) } \overline{U_1 g_{j_1}^{\varepsilon_{j_1}}} = \overline{\overline{U_1} g_{j_1}^{\varepsilon_{j_1}}} = \overline{g_{j_1}^{\varepsilon_{j_1}}}$$

$$= s_{K_{j_1}, g_{j_1}}^{\varepsilon_{j_1}}.$$

Likewise, for $n \geq r+1$, $K_{l_n} = \overline{U_1 g_{l_{r+1}}^{\varepsilon_{l_{r+1}}} \ldots g_{l_{n-1}}^{\varepsilon_{l_{n-1}}}} = \overline{g_{l_{r+1}}^{\varepsilon_{l_{r+1}}} \ldots g_{l_{n-1}}^{\varepsilon_{l_{n-1}}}}$, $\varepsilon_{l_n} = 1$

or $\overline{U_1 g_{l_{r+1}}^{\varepsilon_{l_{r+1}}} \ldots g_{l_{n-1}}^{\varepsilon_{l_{n-1}}} g_{l_n}^{-1}} = \overline{g_{l_{r+1}}^{\varepsilon_{l_{r+1}}} \ldots g_{l_{n-1}}^{\varepsilon_{l_{n-1}}} g_{l_n}^{-1}}$, $\varepsilon_{l_n} = -1$

So, $K_{l_n} = K_{j_{n-r}}$, $n \geq r+1$, and therefore, $s_{K_{l_{r+1}}, g_{l_{r+1}}}^{\varepsilon_{l_{r+1}}} \ldots\ldots s_{K_{l_v}, g_{l_v}}^{\varepsilon_{l_v}} = \tau(U_2)$.

Thus $\tau(U_1U_2) = \tau(U_1)\tau(U_2)$. In fact $\tau(U_1U_2)$ contains precisely the same s-symbols as $\tau(U_1)\tau(U_2)$.

Hence τ is a rewriting process. $\Diamond$

Such a rewriting process obtained from a right coset representation is called a Reidemeister rewriting process. Using a Reidemeister rewriting process τ, the presentation for **H** found in Theorem 3 can be simplified.

Theorem 5 (Reidemeister)

Let τ be the Reidemeister rewriting process defined by (18) for a subgroup **H** of a group **G**. If **G** has the presentation (4), then **H** has the presentation

$$\langle s_{K, g_i}, \ldots \mid s_{K, g_i} = \tau(Kg_i\overline{Kg_i}^{-1}), \tau(KR_jK^{-1}) = 1 \rangle \qquad (19)$$

under the mapping $s_{K, g_i} \rightarrow K g_i \overline{K g_i}^{-1}$, where K is an arbitrary representative (used in the right coset function determining τ), g_i is an arbitrary generator of **G** and R_j an arbitrary defining relator in (4).

Proof

We need to show that relations (6) and (7) can be derived from those in (19). Then we can delete any redundant defining relations by (T2) to get the presentation in Theorem 3.

To simplify the set of relations in (7) we note that any word w is freely equal to UK, where K is $\overline{w}$ and $U = w K^{-1}$ which defines an element of **H**.

Thus,

$\tau(w R_j w^{-1}) \approx \tau(U.K R_j K^{-1}.U^{-1})$

$= \tau(U)\, \tau(K R_j K^{-1})\, \tau(U^{-1})$ since condition (2) holds with τ and $K R_j K^{-1} \in \overline{R} \subseteq H$.

From the proof on page 18 of the corollary to Theorem 4 it can be seen that

$\tau(UU^{-1})$ contains exactly the same s-symbols as $\tau(U)\tau(U^{-1})$. Thus, since $\tau(UU^{-1}) \approx \tau(1) = 1$, $\tau(U^{-1})$ must contain precisely the same s-symbols as $\tau(U)^{-1}$. Therefore, $\tau(wR_jw^{-1})$ is derivable from the relator $\tau(U)\tau(KR_jK^{-1})(\tau(U))^{-1}$ and hence from the relation $\tau(KR_jK^{-1}) = 1$.

We have already shown that the words $Kg_i\overline{Kg_i}^{-1}$ generate the subgroup **H** [Theorem 4]. Thus, the relations $s_{K,g_i} = \tau(Kg_i\overline{Kg_i}^{-1})$ are seen to be the relations in (6), with s_{K,g_i} the defining symbol h_k and $Kg_i\overline{Kg_i}^{-1}$ the subgroup generator h_k written in terms of the group generators g_i.

Obviously K is a word in the g_i so $\{KR_jK^{-1}\} \subseteq \{wR_jw^{-1}\}$.

Let $\mathbf{R}_K = \{\tau(KR_jK^{-1})\}$ and $\mathbf{R}_w = \{\tau(wR_jw^{-1})\}$. Then clearly, $\mathbf{R}_K \subseteq \mathbf{R}_w$ so

$$\langle s_{K,g_i} \mid s_{K,g_i} = \tau(Kg_i\overline{Kg_i}^{-1}), \mathbf{R}_w \rangle \leq \langle s_{K,g_i} \mid s_{K,g} = \tau(Kg_i\overline{Kg_i}^{-1}), \mathbf{R}_K \rangle$$

i.e. $\mathbf{H}_w$ is a factor group of $\mathbf{H}_K$ where $\mathbf{H}_w$ is the presentation for **H** defined in Theorem 3.

However, since the relators $\mathbf{R}_w$ can all be derived from $\mathbf{R}_K$, $\mathbf{H}_K = \mathbf{H}_w$. Therefore, (19) is a presentation for **H**. ◊

The following is an immediate consequence of Theorem 5.

Corollary

If **G** is finitely presented and **H** is of finite index in **G**, then **H** is finitely presented.

It is possible to simplify the presentation (19) still further by restricting ourselves to a special class of right coset functions. A Schreier right coset function is one for which any initial segment of a representative is itself a representative, and the resulting system of coset representatives is called a Schreier system. A Reidemeister rewriting process using a Schreier system is called a Reidemeister-Schreier rewriting process.

It can be shown that there is always a Schreier system of representatives for **G** mod **H**:

Define the length of a coset of **G** mod **H** to be the length of the shortest word in it. Choose the empty word as the representative of **H**, the coset of length zero. If S_1 is a coset of length one, choose any word of length one as its representative. If S_2 has length two, select any word g_1g_2 of length two in S_2. Now, $\overline{g_1}\, g_2$ is also in S_2 by (14) and $\overline{g_1}$ has length at most one, so $\overline{g_1}\, g_2$ has length at most two. Choose $\overline{g_1}\, g_2$ as the representative of S_2. Then, in general, assume we have chosen representatives for all cosets of length < r and if S_r is a coset of length r and $g_1 \cdots g_{r-1}\, g_r$ is a word in S_r we choose $\overline{g_1\, g_2 \cdots g_{r-1}}\, g_r$ (which has length at most r) as a representative of S_r. Thus, if the last symbol is deleted from a representative, we obtain another representative so we have a Schreier system. ◊

Theorem 6 (Schreier)

Let **G** be presented as in (4) and **H** be a subgroup of **G**. If τ is a Reidemeister-Schreier rewriting process, then **H** can be presented as

$$\left\langle s_{K,g_i}, \ldots \mid s_{M,g_{i_r}} = 1,\ \tau(KR_jK^{-1}) = 1 \right\rangle \tag{20}$$

where K is an arbitrary Schreier representative, g_i is an arbitrary generator and R_j is an arbitrary defining relator in (4), and M is a Schreier representative and g_{i_r} a generator such that

$$Mg_{i_r} \approx \overline{Mg_{i_r}}\,.$$

Proof

$Mg_{i_r} \approx \overline{Mg_{i_r}}$ so $Mg_{i_r}\overline{Mg_{i_r}}^{-1} \approx 1$. Hence, by condition (1) of a rewriting process, $\tau(Mg_{i_r}\overline{Mg_{i_r}}^{-1}) \approx \tau(1) = 1$. Thus, the relation $s_{M,g_{i_r}} = \tau(Mg_{i_r}\overline{Mg_{i_r}}^{-1})$ is derivable from the relation

$$s_{M,g_{i_r}} = 1. \tag{21}$$

If $Kg_i \neq \overline{Kg_i}$ then each s-symbol replacing a g-symbol of K in $\tau(Kg\overline{Kg}^{-1})$ will have the form $s_{N,g_{i_j}}$ or $s^{-1}_{Ng_{i_j}^{-1},g_{i_j}}$ where N is the initial segment of K preceding the g-symbol replaced. In the first case, Ng_{i_j} is an initial segment of K and so $\overline{Ng_{i_j}} = Ng_{i_j}$. In the second case, N is an initial segment of K, so $\overline{Ng_{i_j}^{-1}\cdot g_{i_j}} = \overline{N} = N \approx Ng_{i_j}^{-1}\cdot g_{i_j}$.

Thus, for any S-symbol $s_{N, g_{ij}}$ obtained from the substitution letter by letter of the word representing K, we have $\overline{Ng_{ij}} \approx Ng_{ij}$. Using the fact that $\tau(U^{-1})$ contains precisely the same S-symbols as $\tau(U)^{-1}$, $[\tau(\overline{Kg}\, g^{-1}K^{-1})]^{-1}$ contains precisely the same S-symbols as $\tau(Kg\overline{Kg}^{-1})$. Thus, the string of S-symbols corresponding to $\overline{Kg}$ in $\tau(\overline{Kg}\, g^{-1}K^{-1})$ is the inverse of the same string corresponding to $\overline{Kg}^{-1}$ in $\tau(Kg\overline{Kg}^{-1})$. Since $\overline{Kg}$ is a coset representative, we can treat it the same way as K in the paragraph above. Thus, for any S-symbol $s_{N, g_{ij}}$ obtained from the substitution letter by letter of the word representing $\overline{Kg}$, we have $\overline{Ng_{ij}} \approx Ng_{ij}$. Hence, $\tau(Kg\overline{Kg}^{-1}) \approx Kg\overline{Kg}^{-1} = s_{K, g}$ so the relations in (6) are derivable from $s_{K, g} = s_{K, g}$ which is clearly trivial and can be deleted. ◊

I will now describe the computation for an easy example, where the presentation for **H** can be deduced easily by inspection.

$$\mathbf{G} = \langle x, y \mid x^3 = y^3 = (xy)^2 = 1 \rangle$$

$$\mathbf{H} = \langle x \rangle$$

This is Example 1 in Section 1.2, page 5. Referring back to this we can choose coset representatives from the definition of the coset numbers.

$$1y = 2$$
$$2x = 3$$
$$3x = 4$$

1, the empty word, is the coset representative for **H**, coset number 1. Now, the representative of coset 2 is $\overline{1}y = 1y = y$ and the representative of coset 3 is $\overline{2}x$, that is yx. Likewise, the representative of coset 4 is $\overline{3}x = yx.x = yx^2$. Thus, our transversal is

$$K = \{ 1, y, yx, yx^2 \}$$

From our method of construction of the coset representatives it is obvious that we end up with a Schreier system. First of all we find out for which subgroup generators $s_{K, g} = 1$, i.e. $Kg \approx \overline{Kg}$ from the coset table.

	Kx	≈	$\overline{Kx}$	Ky	≈	$\overline{Ky}$
x	x	×	$\overline{x^2}=1$	y	✓	y
y	yx	✓	yx	y^2	×	$\overline{y^2}=yx$
yx	yx^2	✓	yx^2	yxy	×	1
yx^2	yx^3	×	y	yx^2y	×	yx^2

Now from the coset table on page 6 we can work out the coset representative of any word by tracing through the word with coset 1 and looking up the representative of the coset number we end up with.

E.g.

x	y	x	
1	1	2	3

so xyx lies in coset 3 i.e. $\overline{xyx}=yx$. Thus, we have five non-trivial generators, that is, five in which $Kg \neq \overline{Kg}$. The other three give the relations s_{M,g_i} in (20).

We have the following Schreier generators written as s-symbols :

$$s_{1,x} \qquad s_{1,y}$$

$$s_{y,x} \qquad s_{y,y}$$

$$s_{yx,x} \qquad s_{yx,y}$$

$$s_{yx^2,x} \qquad s_{yx^2,y}$$

It can be seen from the above table that $s_{y,x}=1$, $s_{yx,x}=1$ and $s_{1,y}=1$. These are the first set of relations in (20).

Now we can use the three group relators $R_1=x^3$, $R_2=y^3$, $R_1=(xy)^2$ to obtain the subgroup relations $\tau(KR_jK^{-1})$.

R_1

$\tau(1.x^3.1^{-1}) = \tau(x^3) = 1x\bar{x}^{-1}.\bar{x}x\overline{x^2}^{-1}.\overline{x^2}x\overline{x^3}^{-1}$

$= s_{1,x}s_{1,x}s_{1,x}$ since $H=\langle x \rangle$.

$\therefore\ \tau(1.x^3.1^{-1})=1 \Rightarrow (s_{1,x})^3=1.$ *(1)*

$\tau(y.x^3.y^{-1}) = 1y\bar{y}^{-1}.\bar{y}x\overline{yx}^{-1}.\overline{yx}x\overline{yx^2}^{-1}.\overline{yx^2}x\overline{yx^3}^{-1}.\overline{yx^3}y^{-1}\overline{yx^3y^{-1}}^{-1}$

$$= s_{1,y}\, s_{y,x}\, s_{yx,x}\, s_{yx^2,x}\, s_{1,y}^{-1}\,.$$

$$\therefore\ \tau(y.x^3.y^{-1}) = 1 \Rightarrow s_{y,x}\, s_{yx,x}\, s_{yx^2,x} = 1\ . \qquad (2)$$

$\tau(yx.x^3.x^{-1}y^{-1}) \approx \tau(yx^3y^{-1}) \Rightarrow$ the same as relator *(2)*.

$\tau(yx^2.x^3.x^{-2}y^{-1}) \approx \tau(yx^3y^{-1}) \Rightarrow$ the same as relator *(2)*.

R_2

$$\tau(1.y^3.1^{-1}) = \tau(y^3) = 1\, y\, \bar{y}^{-1}.\, \bar{y}\, y\, \overline{y^2}^{-1}.\, \overline{y^2}\, y\, \overline{y^3}^{-1}$$

$$= s_{1,y}\, s_{y,y}\, s_{yx,y}\,.$$

$$\therefore\ \tau(1.y^3.1^{-1}) = 1 \Rightarrow s_{1,y}\, s_{y,y}\, s_{yx,y} = 1. \qquad (3)$$

$\tau(y.y^3.y^{-1}) \approx \tau(y^3) \Rightarrow$ the same as relator *(3)*.

$$\tau(yx.y^3.x^{-1}y^{-1}) = 1\, y\, \bar{y}^{-1}.\, \bar{y}\, x\, \overline{yx}^{-1}.\, \overline{yx}\, y\, \overline{yxy}^{-1}.\, \overline{yxy}\, y\, \overline{yxy^2}^{-1}.\, \overline{yxy^2}\, y\, \overline{yxy^3}^{-1}.$$

$$\overline{yxy^3}\, x^{-1}\, \overline{yxy^3x^{-1}}^{-1}.\, \overline{yxy^3x^{-1}}\, y^{-1}\, \overline{yxy^3x^{-1}y^{-1}}$$

$$= s_{1,y}\, s_{y,x}\, s_{yx,y}\, s_{1,y}\, s_{y,y}\, s_{y,x}^{-1}\, s_{1,y}^{-1}\,.$$

$\therefore\ \tau(yx.y^3.x^{-1}y^{-1}) = 1 \Rightarrow s_{yx,y}\, s_{1,y}\, s_{y,y} = 1$ - a cyclic permutation of *(3)*.

$$\tau(yx^2.y^3.x^{-2}y^{-1}) = 1\, y\, \bar{y}^{-1}.\, \bar{y}\, x\, \overline{yx}^{-1}.\, \overline{yx}\, x\, \overline{yx^2}^{-1}.\, \overline{yx^2}\, y\, \overline{yx^2y}^{-1}.$$

$$\overline{yx^2y}\, y\, \overline{yx^2y^2}^{-1}.\, \overline{yx^2y^2}\, y\, \overline{yx^2y^3}^{-1}.\, \overline{yx^2y^3}\, x^{-1}\, \overline{yx^2y^3x^{-1}}^{-1}.$$

$$\overline{yx^2y^3x^{-1}}\, x^{-1}\, \overline{yx^2y^3x^{-2}}^{-1}.\, \overline{yx^2y^3x^{-2}}\, y^{-1}\, \overline{yx^2y^3x^{-2}y^{-1}}^{-1}$$

$$= s_{1,y}\, s_{y,x}\, s_{yx,x}\, s_{yx^2,y}\, s_{yx^2,y}\, s_{yx^2,y}\, s_{yx,x}^{-1}\, s_{y,x}^{-1}\, s_{1,y}^{-1}$$

$$\therefore\ \tau(yx^2.y^3.x^{-2}y^{-1}) = 1 \Rightarrow (s_{yx^2,y})^3 = 1. \qquad (4)$$

R_3

$$\tau(1.xyxy.1^{-1}) = \tau(xyxy) = 1\, x\, \bar{x}^{-1}.\, \bar{x}\, y\, \overline{xy}^{-1}.\, \overline{xy}\, x\, \overline{xyx}^{-1}.\, \overline{xyx}\, y\, \overline{xyxy}^{-1}$$

$$= s_{1,x}\, s_{1,y}\, s_{y,x}\, s_{yx,y}\,.$$

$$\therefore\ \tau(1.xyxy.1^{-1}) = 1 \Rightarrow s_{1,x}\, s_{1,y}\, s_{y,x}\, s_{yx,y} = 1. \qquad (5)$$

$$\tau(y.xyxy.y^{-1}) \approx \tau(yxyx) = 1\, y\, \bar{y}^{-1}.\, \bar{y}\, x\, \overline{yx}^{-1}.\, \overline{yx}\, y\, \overline{yxy}^{-1}.\, \overline{yxy}\, x\, \overline{yxyx}^{-1}$$

$$= s_{1,y}\, s_{y,x}\, s_{yx,y}\, s_{1,x}\,.$$

$\therefore\ \tau(y.xyxy.y^{-1}) = 1 \Rightarrow s_{1,y}\, s_{y,x}\, s_{yx,y}\, s_{1,x} = 1$ - a cyclic permutation of *(5)*.

$$\tau(yx.xyxy.x^{-1}y^{-1}) = 1\,y\bar{y}^{-1}.\,\bar{y}\,x\,\overline{yx}^{-1}.\,\overline{yx}\,x\,\overline{yx^2}^{-1}.\,\overline{yx^2}\,y\,\overline{yx^2y}^{-1}.$$
$$\overline{yx^2y}\,x\,\overline{yx^2yx}^{-1}.\,\overline{yx^2yx}\,y\,\overline{yx^2yxy}^{-1}.\,\overline{yx^2yxy}\,x^{-1}\,\overline{yx^2yxyx^{-1}}^{-1}.$$
$$\overline{yx^2yxyx^{-1}}\,y^{-1}\,\overline{yx^2yxyx^{-1}y^{-1}}^{-1}$$
$$= s_{1,y}\, s_{y,x}\, s_{yx,x}\, s_{yx^2,y}\, s_{yx^2,x}\, s_{y,y}\, s_{y,x}^{-1}\, s_{1,y}^{-1}.$$

$$\therefore\ \tau(yx.xyxy.x^{-1}y^{-1}) = 1 \Rightarrow s_{yx,x}\, s_{yx^2,y}\, s_{yx^2,x}\, s_{y,y} = 1. \qquad (6)$$

$$\tau(yx^2.xyxy.x^{-2}y^{-1}) = 1\,y\bar{y}^{-1}.\,\bar{y}\,x\,\overline{yx}^{-1}.\,\overline{yx}\,x\,\overline{yx^2}^{-1}.\,\overline{yx^2}\,x\,\overline{yx^3}^{-1}.\,\overline{yx^3}\,y\,\overline{yx^3y}^{-1}.$$
$$\overline{yx^3y}\,x\,\overline{yx^3yx}^{-1}.\,\overline{yx^3yx}\,y\,\overline{yx^3yxy}^{-1}.\,\overline{yx^3yxy}\,x^{-1}\,\overline{yx^3yxyx^{-1}}^{-1}.$$
$$\overline{yx^3yxyx^{-1}}\,x^{-1}\,\overline{yx^3yxyx^{-2}}^{-1}.\,\overline{yx^3yxyx^{-2}}\,y^{-1}\,\overline{yx^3yxyx^{-2}y^{-1}}^{-1}$$
$$= s_{1,y}\, s_{y,x}\, s_{yx,x}\, s_{yx^2,x}\, s_{y,y}\, s_{yx,y}\, s_{yx^2,y}\, s_{yx,x}^{-1}\, s_{y,x}^{-1}\, s_{1,y}^{-1}.$$

$\therefore\ \tau(yx^2.xyxy.x^{-2}y^{-1}) = 1 \Rightarrow s_{yx^2,x}\, s_{y,y}\, s_{yx,y}\, s_{yx^2,y} = 1$ - a cyclic permutation of relation *(6)*.

Now we can write out our presentation for **H** as given by (20) omitting duplicate relations :

$$\mathbf{H} = \langle s_{1,x}, s_{1,y}, s_{y,x}, s_{y,y}, s_{yx,x}, s_{yx,y}, s_{yx^2,x}, s_{yx^2,y} \mid s_{y,x} = 1,\ s_{yx,x} = 1,$$
$$s_{1,y} = 1,\ (s_{1,x})^3 = 1,\ s_{y,x}\, s_{yx,x}\, s_{yx^2,x} = 1,\ \ s_{1,y}\, s_{y,y}\, s_{yx,y} = 1,$$
$$(s_{yx^2,y})^3 = 1,\ s_{1,x}\, s_{1,y}\, s_{y,x}\, s_{yx,y} = 1,\ \ s_{yx,x}\, s_{yx^2,y}\, s_{yx^2,x}\, s_{y,y} = 1 \rangle.$$

Using Tietze transformations (T4)' we can remove the trivial generators $s_{y,x}$, $s_{yx,x}$ and $s_{1,y}$ to obtain the following presentation :

$$\mathbf{H} = \langle s_{1,x}, s_{y,y}, s_{yx,y}, s_{yx^2,x}, s_{yx^2,y} \mid (s_{1,x})^3 = 1,\ s_{yx^2,x} = 1,\ s_{y,y}\, s_{yx,y} = 1,$$
$$(s_{yx^2,y})^3 = 1,\ s_{1,x}\, s_{yx,y} = 1,\ \ s_{yx^2,y}\, s_{yx^2,x}\, s_{y,y} = 1 \rangle. \qquad (21)$$

Now $s_{yx^2,x}$ is trivial so it can also be removed. From the third relation $s_{yx,y} = s_{y,y}^{-1}$ thus, using (T4)', $s_{yx,y}$ can be eliminated. From the fifth relation, $s_{1,x}\, s_{yx,y} = 1$ so $s_{1,x} = s_{y,y}$ and $s_{y,y}$ can be eliminated. From the sixth relation $s_{yx^2,y}\, s_{y,y} = 1$ so $s_{yx^2,y} = s_{1,x}$. This then gives us :

$\mathbf{H} = \langle s_{1,x} \mid (s_{1,x})^3 = 1 \rangle$ - the cyclic group of order 3 as expected.

As you can see this is a very long-winded process. By hand the procedure is too complicated to carry out for anything but the simplest examples.

A slightly different way of doing this is to keep all calculations in terms of the original subgroup generators until the end when they can be translated into the s-symbols. This makes it easier to calculate $\tau(KR_jK^{-1})$ since, if $KR_jK^{-1} = g_{i_1}g_{i_2}\ldots g_{i_n}$ then $\tau(KR_jK^{-1}) \approx g_{i_1}g_{i_2}\cdots g_{i_n}\overline{g_{i_1}g_{i_2}\cdots g_{i_n}}^{-1} = g_{i_1}g_{i_2}\cdots g_{i_n}$, since $g_{i_1}g_{i_2}\ldots g_{i_n}$ is a relation and so has coset representative 1.

Using the table on page 24 we calculate $Kg\overline{Kg}^{-1}$ and $\tau(KR_jK^{-1})$.

	$Kg\overline{Kg}^{-1}$		$\tau(KR_jK^{-1})$		
K	x	y	R_1	R_2	R_3
1	x	1	x^3	y^3	xyxy
y	1	$y^2x^{-1}y^{-1}$	yx^3y^{-1}	y^3	yxyx
yx	1	yxy	yx^3y^{-1}	$yxy^3x^{-1}y^{-1}$	$yx^2yxyx^{-1}y^{-1}$
yx^2	yx^3y^{-1}	$yx^2yx^{-2}y^{-1}$	yx^3y^{-1}	$yx^2y^3x^{-2}y^{-1}$	$yx^3yxyx^{-2}y^{-1}$

Now we can translate these generators and relations into terms of s-symbols.

$$s_{1,x} = x$$

$$s_{y,y} = y^2x^{-1}y^{-1}$$

$$s_{yx,y} = yxy$$

$$s_{yx^2,x} = yx^3y^{-1}$$

$$s_{yx^2,y} = yx^2yx^{-2}y^{-1}$$

By inspection the R_1 relations then become $(s_{1,x})^3 = 1$ and $s_{yx^2,x} = 1$.

R_2 relations : $s_{y,y}\, s_{yx,y} = 1$, $s_{yx,y}\, s_{y,y} = 1$, $(s_{yx^2,y})^3 = 1$.

R_3 relations : $s_{1,x}\, s_{yx,y} = 1$, $s_{yx,y}\, s_{1,x} = 1$, $s_{yx^2,y}\, s_{yx^2,x}\, s_{y,y} = 1$.

This gives us exactly the same presentation as in (21). Note that it is not always such a simple process to convert these relations into terms of the s-symbols.

§1.5 The Modified Todd-Coxeter Algorithm

In the last section I showed how a presentation for a subgroup, **H**, of a group **G**, could be constructed using a rewriting process called Reidemeister-Schreier. This method resulted in a presentation on a set of Schreier generators, not the original subgroup generators, h_k, of **H**. Such a presentation contains a large number of Schreier generators, many of which prove to be redundant.

There is another process, which I shall call the Modified Todd-Coxeter Algorithm, which does find a presentation for **H** in terms of the given subgroup generators, h_k. For this method, most of the calculation is done in parallel with the coset enumeration; however in the coset table we actually keep track of the action of coset representatives on the generators rather than the coset numbers. Thus, information of the form $\alpha g_i = \beta$ with α and β two coset numbers now becomes $\alpha g_i = v_{\alpha, g_i}(h_k).\beta$ where α and β are the coset representatives of cosets α and β and $v_{\alpha, g_i}(h_k)$ is a word in the subgroup generators, h_k.

In the late 1960's and the 1970's a number of mathematicians such as Leech [25], Benson and Mendelsohn [5], McLain [26], Mendelsohn ([28],[29]), Beetham and Campbell [4] gave procedures for finding subgroup relations and presentations for the subgroup in terms of the given subgroup generators, h_k. These are all slightly different. Here I will give a description of the process which most closely follows that given by Neubüser in a survey article [30].

Method

Let $\mathbf{H} = \langle h_1, h_2, \ldots, h_s \rangle$ be a subgroup of **G** presented as in (4). Then $h_k = J_k(g_i)$ where $J_k(g_i)$ denotes the expression of h_k as a word in the generators g_i of **G**.

Carrying out the Todd-Coxeter method as described in Section 1.2, we first trace through the subgroup generator tables with coset 1. As we go, we build up an "augmented coset table" which records the information $\alpha g_i = v_{\alpha, g_i}(h).\beta$. Each definition that we make gives us coset table entries of the form $\beta g_i = \gamma$ and $\gamma g_i^{-1} = \beta$, which tell us that coset representative β multiplied on the right by group generator g_i gives coset representative γ and its inverse equation [i.e. in the case of a definition, v(h)=1, the empty word].

In the rest of this description I will write equations of the form $\alpha g_i = v_{\alpha, g_i}(h).\beta$ as $\bar{\alpha} g_i = v_{\alpha, g_i}(h).\bar{\beta}$, so that it is obvious that I am regarding α and β as coset representatives, not coset numbers. However, in the tables, I will omit the bar as it makes them clearer to read.

In the subgroup tables we can no longer use the fact that $1 J_k(g_i) = 1$ but rather that $\bar{1} J_k(g_i) = h_k.\bar{1}$ where h_k is the generating symbol for the kth subgroup generator, $J_k(g_i)$. Thus, at the end of the row in the kth subgroup generator table we insert $h_k.1$ instead of 1. When a row of the a table closes we get either

(i) a deduction

(ii) no new information

(iii) a coincidence

as explained on page 4.

(i) If $\alpha g_i = \beta$ is a deduction from a subgroup generator table where g_i is a generator such that $w_1 g_i w_2 = h_k$, with $w_1 = g_{j_1} \ldots g_{j_p}$ and $w_2 = g_{k_1} \ldots g_{k_q}$, then we have the following subgroup generator table.

	$\leftarrow w_1(g) \rightarrow$		$\leftarrow w_2(g) \rightarrow$	
	$g_{j_1} \ldots\ldots g_{j_p}$	g_i	$g_{k_1} \ldots\ldots g_{k_q}$	
1	$\ldots\ldots\ldots \alpha$	β	$\ldots\ldots\ldots$	$h_k.1$

Tracing along the row in the forward direction we look up $\bar{1} g_{j_1}$ in the augmented coset table, $\bar{1} g_{j_1} = v_{1, g_{j_1}}(h).\bar{\gamma}$ say. Then we look up $\bar{\gamma} g_{j_2}$ which is $v_{\gamma, g_{j_2}}(h).\bar{\delta}$, say. We concatenate the words in h i.e. we have $v_{1, g_{j_1}}(h) v_{\gamma, g_{j_2}}(h)$ so far. In this way the h-words are built up until we reach

$$\bar{\xi} g_{j_p} = v_{1, g_{j_1}}(h) v_{\gamma, g_{j_2}}(h) \ldots\ldots v_{\xi, g_{j_p}}(h).\bar{\alpha}$$

[i.e. $\bar{1} w_1 = v_{1, g_{j_1}}(h) v_{\gamma, g_{j_2}}(h) \ldots\ldots v_{\xi, g_{j_p}}(h).\bar{\alpha}$]. Similarly, from the other end, $\bar{1} g_{k_q}^{-1}$ is checked in the coset table and h_k concatenated with the corresponding h-word for $\bar{1} g_{k_q}^{-1}$. Again this process is continued until $\bar{\psi} g_{k_1}^{-1} = h_k v_{1, g_{k_q}^{-1}}(h) \ldots\ldots v_{\psi, g_{k_1}^{-1}}(h).\bar{\beta}$ is reached. [i.e. $h_k.\bar{1} w_2^{-1} = h_k v_{1, g_{k_q}^{-1}}(h) \ldots\ldots v_{\psi, g_{k_1}^{-1}}(h).\bar{\beta}$.]

Thus, $\bar{\alpha} g_i = \bar{\alpha} w_1^{-1} h_k w_2^{-1} = (v_{1, g_{j_1}}(h) v_{\gamma, g_{j_2}}(h) \ldots\ldots v_{\xi, g_{j_p}}(h))^{-1}.\bar{1} h_k w_2^{-1}$

$= (v_{1, g_{j_1}}(h) v_{\gamma, g_{j_2}}(h) \ldots\ldots v_{\xi, g_{j_p}}(h))^{-1} h_k.\bar{1} w_2^{-1}$

$$= (v_{1,g_{j_1}}(h)\, v_{\gamma,g_{j_2}}(h) \ldots\ldots v_{\xi,g_{j_p}}(h))^{-1}\, h_k\, v_{1,g_{k_q}^{-1}}(h) \ldots\ldots v_{\psi,g_{k_1}^{-1}}(h).\bar{\beta}\ .$$

Then, the word $(v_{1,g_{j_1}}(h)\, v_{\gamma,g_{j_2}}(h) \ldots\ldots v_{\xi,g_{j_p}}(h))^{-1}\, h_k\, v_{1,g_{k_q}^{-1}}(h) \ldots\ldots v_{\psi,g_{k_1}^{-1}}(h)$ is placed in the augmented coset table as the v(h) entry for $\bar{\alpha}\, g_i$.

[When the h-words are concatenated, the resulting word can be freely reduced to get rid of any generator adjacent to its inverse.]

Similarly, if $\alpha g_i = \beta$ is a deduction gained from a relation table then we have a relator R_j where $R_j = w_1\, g_i\, w_2$, with w_1 and w_2 words in $X^{\pm 1}$ or the empty word, 1, and $\bar{\alpha}\, g_i$ undefined in the augmented coset table. If the deduction occurs in the λth row we have $\bar{\lambda} w_1(g) = V_{\lambda,w_1}(h).\bar{\alpha}$ and $\bar{\lambda} w_2^{-1}(g) = V_{\lambda,w_2^{-1}}(h).\bar{\beta}$ where $V_{\lambda,w_1}(h)$ and $V_{\lambda,w_2^{-1}}(h)$ are words in the h-symbols computed inductively by concatenating the augmented coset table words v_{γ,g_i} from $\bar{\lambda} w_1$ and $\bar{\lambda} w_2^{-1}$ as above.

	$\leftarrow\ w_1(g)\ \rightarrow$		$\leftarrow\ w_2(g)\ \rightarrow$	
	$g_{j_1} \ldots\ldots g_{j_p}$	g_i	$g_{k_1} \ldots\ldots g_{k_q}$	
λ	$\ldots\ldots\ldots\ \alpha$		$\beta\ \ldots\ldots\ldots$	λ

In this case, $\bar{\alpha}\, g_i = \bar{\alpha}\, w_1^{-1} w_2^{-1} = V_{\lambda,w_1}^{-1}(h)\, V_{\lambda,w_2^{-1}}(h).\bar{\beta}$

i.e. $v_{\alpha,g_i}(h) = V_{\lambda,w_1}^{-1}(h)\, V_{\lambda,w_2^{-1}}(h)$.

Thus, $V_{\lambda,w_1}^{-1}(h)\, V_{\lambda,w_2^{-1}}(h).\bar{\beta}$ is placed in the augmented coset table.

(ii) If, in tracing out the kth subgroup generator, no new cosets are defined, and, when the row closes, no new information is found, then we obtain a relation between the generators of **H** of the form $h_k = V_{1,h_k(g_i)}(h)$ where the word $V_{1,h_k(g_i)}(h)$ is found inductively by concatenating the augmented coset table words obtained by tracing out $\bar{1} J_k(g_i)$ as described above. This shows that the generator h_k is redundant and it will not become incorporated into the augmented coset table.

Similarly, a relation between the subgroup generators is also obtained if a row of a relation table closes in this way. Here, when we trace out and concatenate the h-words, we obtain a final equation of the form :

$$\bar{\lambda} R_j = V_{\lambda, R_j}(h) . \bar{\lambda} .$$

Since $R_j=1$, clearly $\bar{\lambda} 1_G = 1_H . \bar{\lambda}$ and so $V_{\lambda, R_j}(h)=1$ is a relation in **H**.

Usually nothing is done about these relations until the coset table closes, then all the subgroup relations are computed at the same time.

(iii) If a row closes with the new information $\alpha g_i = \beta$ leading to a coincidence $\beta \equiv \gamma$ with $\beta < \gamma$ we have

$$\bar{\alpha} g_i = v'_{\alpha, g_i}(h) . \bar{\beta}$$

$$\bar{\alpha} g_i = v_{\alpha, g_i}(h) . \bar{\gamma}$$

where $v'_{\alpha, g_i}(h)$ is computed inductively as in (i) above and $v_{\alpha, g_i}(h)$ is already present in the augmented coset table. Thus,

$$v'_{\alpha, g_i}(h) . \bar{\beta} = v_{\alpha, g_i}(h) . \bar{\gamma} \Rightarrow v_{\alpha, g_i}(h)^{-1} v'_{\alpha, g_i}(h) . \bar{\beta} = \bar{\gamma}.$$

We then substitute $v_{\alpha, g_i}(h)^{-1} v'_{\alpha, g_i}(h) . \bar{\beta}$ for $\bar{\gamma}$ throughout the augmented coset table, and replace γ by β in the relation and subgroup generator tables. If any of these substitutions leads to another coincidence it is dealt with in the same way.

When all of the tables close we can use the augmented coset table to find subgroup relations as follows :

Tracing through each relator with each coset representative $\bar{\lambda}$ in turn, concatenating h-words as we go, we get

$$\bar{\lambda} R_j = V_{\lambda, R_j}(h) . \bar{\lambda} \quad 1 \le j \le t, \, 1 \le \lambda \le |\mathbf{G} : \mathbf{H}|, \text{ implying}$$

$$V_{\lambda, R_j}(h) = 1. \tag{22}$$

Then for each subgroup generator $J_k(g_i)$ we can trace through with coset 1, concatenating the h-words to obtain

$$\bar{1} J_k(g_i) = V_{1, J_k}(h) . \bar{1}, \quad 1 \le k \le s, \text{ implying}$$

$$h_k = w_{1, h_k(g_i)}(h). \tag{23}$$

These two sets of relations (22) and (23) define the subgroup **H**. This will be proved later in Section 1.5.1, but now I shall demonstrate the method with a familiar example.

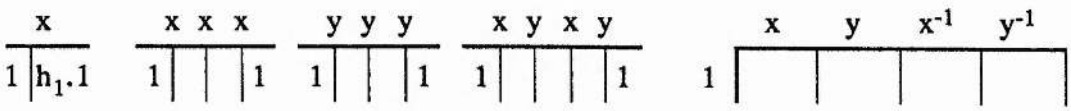
Example 1

$G = \langle x, y \mid x^3 = y^3 = (xy)^2 = 1 \rangle$

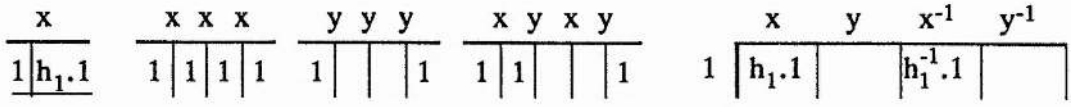
$H = \langle x \rangle$

We set up the tables as follows :

Subgroup Relation Tables Augmented Coset Table



Let x be subgroup generator h_1. Straightaway we gain the deduction $\bar{1}x=h_1.\bar{1}$. This is filled into the augmented coset table and 1x=1 filled into the other tables.



The first row of relation table R_1 closes giving no new information. We will ignore this for the moment, but at the end it will provide us with a valuable relation. Now define $\bar{1}y=\bar{2}$, then $\bar{2}x=\bar{3}$, giving us the deduction 3y=1 from R_3. However, we have to calculate the h-word $v_{3,y}$, before we can enter this information in the augmented coset table.

Here $R_3=(xyx)y$ so $w_1=xyx$, $w_2=1$.

Then $\bar{3}y=\bar{3}(xyx)^{-1}1^{-1}$ from (i) on page 29.

$= \bar{3}x^{-1}y^{-1}x^{-1}$

$= 1.\bar{2}y^{-1}x^{-1}$

$= 1.\bar{1}x^{-1}$

$= h_1^{-1}.\bar{1}$.

Therefore we have $v_{3,y}(h) = h_1^{-1}$ and $v_{1,y^{-1}}(h) = h_1$. Clearly, this combination of coset representative and relation will give us a trivial relation in the final presentation, since we used the information $V_{1,R_3}(h)=1$ in the calculation of the h-word for the deduction.

We then get the deduction 2y=3 from the first row of relation 2. Here $R_2=y^3$

so $w_1 = y$ and $w_2 = y$.

Then, $\bar{2}y = \bar{2}y^{-1}y^{-1}$

$= 1.\bar{1}y^{-1}$

$= h_1.\bar{3}$.

So, $v_{2,y}(h) = h_1$ and $v_{3,y^{-1}}(h) = h_1^{-1}$. Our coset tables are now as follows :

x
1 | $h_1.1$

	x	x	x
1	1	1	1
2	3		2
3		2	3

	y	y	y
1	2	3	1
2	3	1	2
3	1	2	3

	x	y	x	y
1	1	2	3	1
2	3	1	1	2
3			2	3

	x	y	x^{-1}	y^{-1}
1	$h_1.1$	2	$h_1^{-1}.1$	$h_1.3$
2	3	$h_1.3$		1
3		$h_1^{-1}.1$	2	$h_1^{-1}.2$

Now we define $\bar{3}x = \bar{4}$ in the second row of the first relation table, closing the row and giving us the deduction $4x = 2$.

Here, $\bar{2}(x^2)x = \bar{2}$ so $w_1 = x^2$, $w_2 = 1$.

Therefore, $\bar{4}x = \bar{4}x^{-2} = 1.\bar{3}x^{-2} = 1.\bar{2}$. Hence the h-word is just the empty word in this case. The updated tables are as follows :

x
1 | $h_1.1$

	x	x	x
1	1	1	1
2	3	4	2
3	4	2	3
4	2	3	4

	y	y	y
1	2	3	1
2	3	1	2
3	1	2	3
4			4

	x	y	x	y
1	1	2	3	1
2	3	1	1	2
3	4	4	2	3
4	2	3		4

	x	y	x^{-1}	y^{-1}
1	$h_1.1$	2	$h_1^{-1}.1$	$h_1.3$
2	3	$h_1.3$	4	1
3	4	$h_1^{-1}.1$	2	$h_1^{-1}.2$
4	2		3	

We now find that that $4y = 4$ from the third row of relation table 3. So,

$$\bar{4}y = \bar{4}x^{-1}(xy)^{-1} = \bar{4}x^{-1}y^{-1}x^{-1} = 1.\bar{3}y^{-1}x^{-1} = h_1^{-1}.\bar{2}x^{-1} = h_1^{-1}.\bar{4} .$$

Thus, $v_{4,y}(h) = h_1^{-1}$ and $v_{4,y^{-1}}(h) = h_1$. This completes our three sets of tables.

x
1 | $h_1.1$

	x	x	x
1	1	1	1
2	3	4	2
3	4	2	3
4	2	3	4

	y	y	y
1	2	3	1
2	3	1	2
3	1	2	3
4	4	4	4

	x	y	x	y
1	1	2	3	1
2	3	1	1	2
3	4	4	2	3
4	2	3	4	4

	x	y	x^{-1}	y^{-1}
1	$h_1.1$	2	$h_1^{-1}.1$	$h_1.3$
2	3	$h_1.3$	4	1
3	4	$h_1^{-1}.1$	2	$h_1^{-1}.2$
4	2	$h_1^{-1}.4$	3	$h_1.4$

We can now compute the subgroup relations :-

Relations from subgroup generators

$h_1 = h_1$ a trivial relation

Relations from group relations

R_1 $h_1^3 = 1$

R_2 $h_1 h_1^{-1} = 1$ trivial

$h_1^{-3} = 1$

R_3 $h_1 h_1^{-1} = 1$ trivial

$h_1^{-1} h_1 = 1$ trivial

Hence we have the presentation $\langle h_1 \mid h_1^3 = 1 \rangle$, which is $\langle x \mid x^3 = 1 \rangle$ rewriting the generator h_1 in terms of the generators of **G**.

Now we will look at another example, this time one with coincidences:-

Example 2

$\mathbf{G} = \langle x, y \mid y^{-1}x^2y = x^3 \rangle$

$\mathbf{H} = \langle x^2, y \rangle$

Now $h_1 = x^2$ and $h_2 = y$.

Our tables are as follows :-

Subgroup Generators

x	x	y
1	$h_1.1$	1 $h_2.1$

Relators

y^{-1}	x	x	y	x^{-1}	x^{-1}	x^{-1}	
1							1

Augmented Coset Table

	x	y	x^{-1}	y^{-1}
1				

From the second subgroup generator we obtain the deduction $\overline{1}y = h_2.\overline{1}$. Now define $\overline{1}x = \overline{2}$ in the first subgroup generator. This gives us the deduction $2x = 1$. In fact $\overline{2}x = h_1.\overline{1}$, since the only other entry in the line is a definition. On completion of

the first row of the relation table we discover a coincidence. We find that $2x^{-1}= 2$ but that we already have $2x^{-1}= 1$ in the tables. Therefore, $2 \equiv 1$.

x	x	
1	2	$h_1.1$

y	
1	$h_2.1$

y^{-1}	x	x	y	x^{-1}	x^{-1}	x^{-1}	
1	1	2	1	1	2	2	1
2							2

	x	y	x^{-1}	y^{-1}
1	2	$h_2.1$	$h_1^{-1}.2$	$h_2^{-1}.1$
2	$h_1.1$		1	

From the first row of the relation table, $\bar{2}x^{-1} = \bar{2}xy^{-1}x^{-2}yx$

$= h_1.\bar{1}y^{-1}x^{-2}yx$

$= h_1h_2^{-1}.\bar{1}x^{-2}yx$

$= h_1h_2^{-1}h_1^{-1}.\bar{2}x^{-1}yx$

$= h_1h_2^{-1}h_1^{-1}.\bar{1}yx$

$= h_1h_2^{-1}h_1^{-1}h_2.\bar{1}x$

$= h_1h_2^{-1}h_1^{-1}h_2.\bar{2}$.

Therefore, $h_1h_2^{-1}h_1^{-1}h_2.\bar{2} = 1.\bar{1} \Rightarrow \bar{2} = h_2^{-1}h_1h_2h_1^{-1}.\bar{1}$.

We shall call $h_2^{-1}h_1h_2h_1^{-1}$ the coincidence word $w_{2,1}(h)$. Now, replacing 2 by 1 in the subgroup generator and relation tables, and replacing $\bar{2}$ by $h_2^{-1}h_1h_2h_1^{-1}.\bar{1}$ in the augmented coset table, we end up with

x	x	
1	1	$h_1.1$

y	
1	$h_2.1$

y^{-1}	x	x	y	x^{-1}	x^{-1}	x^{-1}	
1	1	1	1	1	1	1	1

	x	y	x^{-1}	y^{-1}
1	$h_2^{-1}h_1h_2h_1^{-1}.1$	$h_2.1$	$h_1^{-1}h_2^{-1}h_1h_2h_1^{-1}.1$	$h_2^{-1}.1$

At this point I shall insist that the h-words on the right-hand side of the augmented coset table are the inverses of the corresponding words on the left, i.e. $v_{1,x^{-1}}$ is changed to $h_1h_2^{-1}h_1^{-1}h_2$. This then gives us the following relations :

Relations from subgroup generators

$$h_1 = h_2^{-1}h_1h_2h_1^{-1}.h_2^{-1}h_1h_2h_1^{-1} \qquad (1)$$

$h_2 = h_2$ a trivial relation

Relations from group relator

$$h_2^{-1}.h_2^{-1}h_1h_2h_1^{-1}.h_2^{-1}h_1h_2h_1^{-1}.h_2.(h_1h_2^{-1}h_1^{-1}h_2)^3=1 \qquad (2)$$

Now *(1)* can be cyclically permuted to give

$$h_1h_2^{-1}h_1^{-1}h_2h_1h_2^{-1}h_1^{-1}h_2h_1=1 \qquad (3)$$

Expanding *(2)* we obtain

$$h_2^{-1}(h_2^{-1}h_1h_2h_1^{-1}h_2^{-1}h_1h_2h_1^{-1})\,h_2h_1h_2^{-1}h_1^{-1}h_2h_1h_2^{-1}h_1^{-1}h_2h_1h_2^{-1}h_1^{-1}h_2=1$$

$\Rightarrow h_2^{-1}(h_1)\,h_2h_1h_2^{-1}h_1^{-1}h_2h_1h_2^{-1}h_1^{-1}h_2h_1h_2^{-1}h_1^{-1}h_2=1$ using *(1)*

$\Rightarrow h_2^{-1}h_1h_2(h_1h_2^{-1}h_1^{-1}h_2h_1h_2^{-1}h_1^{-1}h_2h_1)\,h_2^{-1}h_1^{-1}h_2=1$

$\Rightarrow h_2^{-1}h_1h_2(1)\,h_2^{-1}h_1^{-1}h_2=1$ using *(3)*

$\Rightarrow 1=1$ freely reducing the relation, which is trivial.

Therefore, *(1)* and *(2)* together contain exactly the same information as *(1)* alone, so *(2)* is redundant.

Hence, $\mathbf{H} = \langle h_1, h_2 \mid h_1=(h_2^{-1}h_1h_2h_1^{-1})^2 \rangle$

$\cong \langle x^2, y \mid x^2=(y^{-1}x^2yx^{-2})^2 \rangle$.

This example is particularly interesting since we have complete collapse. This means that we have found a presentation for the original group on a new set of generators. Thus, using the modified Todd-Coxeter algorithm, it is possible to find a presentation on any set of words in the elements of $\mathbf{X}^{\pm 1}$ which generate **G**.

§1.5.1 Proof of Modified Todd-Coxeter Process

Now that we have shown the technique in action, we need to prove that the resulting presentation is sufficient to define **H**. To do this, I will first show that the method, as described, defines a rewriting process for **H**, which I shall call τ_m. Note that the condition I imposed on the augmented coset table above, i.e. that $v_{\lambda,g_i}(h)$ is exactly the same word as $v_{\mu,g_i^{-1}}(h)$ if $\lambda g_i = \mu$, is not necessary for the validity of the final presentation obtained. However, this condition always holds for the augmented

coset tables produced by the programs described in Chapters 2 and 3, and I have chosen to give a proof of the process which depends on this fact. Other proofs which use the complete table can be found in [4], [26] and [30].

Let $U(g_i)$ be an element of **H** and let $\tau_m(U(g_i)) = V_{1,U(g_i)}(h)$ if $\bar{1}U(g_i) = V_{1,U(g_i)}(h).\bar{1}$ where $V_{1,U(g_i)}(h)$ is obtained by tracing through $U(g_i)$ with coset representative $\bar{1}$, concatenating the h-words from the augmented coset table.

τ_m satisfies conditions (1) and (2) of a rewriting process.

(1) Let $U(g_i) \approx U^*(g_i)$. If there exists a generator or its inverse $g_p \in \mathbf{X}^{\pm 1}$ such that $U(g_i) = w_1(g_i)w_2(g_i)$ and $U^*(g_i) = w_1(g_i)g_p g_p^{-1} w_2(g_i)$ then

$$\bar{1}w_1(g_i) = V_{1,w_1(g_i)}(h).\bar{\lambda} \text{ where } \bar{\lambda} \text{ is some coset representative}$$

and $\bar{1}w_1(g_i)w_2(g_i) = V_{1,w_1(g_i)}(h).\bar{\lambda}w_2(g_i) = V_{1,w_1(g_i)}(h)\,V_{\lambda,w_2(g_i)}(h).\bar{1}$ since $w_1(g_i)w_2(g_i) \in \mathbf{H}$.

$$\therefore \tau_m(w_1(g_i)w_2(g_i)) = V_{1,w_1(g_i)}(h)\,V_{\lambda,w_2(g_i)}(h).$$

Now $\bar{\lambda}\,g_p = v_{\lambda,g_p}(h).\bar{\mu}$ for some coset representative $\bar{\mu}$ and $\bar{\mu}\,g_p^{-1} = v_{\mu,g_p^{-1}}(h).\bar{\lambda}$ from the augmented coset table. However, from the construction of the augmented coset table,

$$v_{\mu,g_p^{-1}}(h) \text{ is exactly the same word as } v_{\lambda,g_p}(h)^{-1} \text{ if } \lambda\,g_p = \mu. \qquad (24)$$

Hence, $\bar{1}w_1(g_i)g_p g_p^{-1} w_2(g_i) = V_{1,w_1(g_i)}(h)\;v_{\lambda,g_p}(h)\,v_{\mu,g_p^{-1}}(h)\,V_{\lambda,w_2(g_i)}(h).\bar{1}$

$$= V_{1,w_1(g_i)}(h)\;v_{\lambda,g_p}(h)\,v_{\lambda,g_p}^{-1}(h)\,V_{\lambda,w_2(g_i)}(h).\bar{1}$$

$$\approx V_{1,w_1(g_i)}(h)\;V_{\lambda,w_2(g_i)}(h).\bar{1}$$

i.e. $\tau_m(w_1(g_i)g_p g_p^{-1} w_2(g_i)) \approx V_{1,w_1(g_i)}(h)\;V_{\lambda,w_2(g_i)}(h) = \tau_m(w_1(g_i)w_2(g_i))$.

Therefore, modulo an induction over the number of trivial relators $g_i^{\varepsilon_i} g_i^{-\varepsilon_i}$ in the two words U and U^*,

$$\tau_m(U(g_i)) \approx \tau_m(U^*(g_i)).$$

(2) $U_1(g_i)$ and $U_2(g_i)$ are elements of **H**. Therefore,

$$\bar{1}U_1(g_i) = V_{1,U_1(g_i)}(h).\bar{1} \text{ and } \bar{1}U_2(g_i) = V_{1,U_2(g_i)}(h).\bar{1}$$

i.e. $\tau_m(U_1(g_i)) = V_{1,U_1(g_i)}(h)$ and $\tau_m(U_2(g_i)) = V_{1,U_2(g_i)}(h)$.

By (1) above, $\tau_m(U_1.U_2) \approx \tau_m((U_1)(U_2))$ where the product $(U_1)(U_2)$ is <u>not</u> freely reduced.

$\bar{1}(U_1)(U_2) = V_{1,U_1(g_i)}(h).\bar{1}\,U_2 = V_{1,U_1(g_i)}(h)\,V_{1,U_2(g_i)}(h).\bar{1}$

$\Rightarrow \tau_m((U_1)(U_2)) = V_{1,U_1(g_i)}(h)\,V_{1,U_2(g_i)}(h) \approx \tau_m(U_1)\,\tau_m(U_2)$

so $\tau_m(U_1.U_2) \approx \tau_m(U_1)\,\tau_m(U_2)$.

Now, do $U(g_i)$ and $V_{1,U(g_i)}(h)$ define the same element of **H** ?

$U(g_i) \in \mathbf{H}$ so can be expressed as a word in the generators $J_k(g_i)$ of **H** and their inverses, i.e.:

$$U(g_i) = J_{k_1}(g_i)^{\varepsilon_{k_1}} J_{k_2}(g_i)^{\varepsilon_{k_2}} \dots J_{k_r}(g_i)^{\varepsilon_{k_r}}.$$

Now,

$$U(g_i)(J_{k_1}(g_i)^{\varepsilon_{k_1}} J_{k_2}(g_i)^{\varepsilon_{k_2}} \dots J_{k_r}(g_i)^{\varepsilon_{k_r}})^{-1} = 1 \qquad (25)$$

so is a relation which holds in **G**. Therefore, it is freely equal to a finite product of conjugates of the defining relators $R_j(g_i)$ $j = 1, \dots, t$.

Hence,

$$U(g_i)(J_{k_1}(g_i)^{\varepsilon_{k_1}} J_{k_2}(g_i)^{\varepsilon_{k_2}} \dots J_{k_r}(g_i)^{\varepsilon_{k_r}})^{-1} \approx w_1(g_i)R_{j_1}w_1^{-1}(g_i) \dots w_p(g_i)R_{j_p}w_p^{-1}(g_i).$$

Now from property (1),

$$\tau_m\big(U(g_i)(J_{k_1}(g_i)^{\varepsilon_{k_1}} J_{k_2}(g_i)^{\varepsilon_{k_2}} \dots J_{k_r}(g_i)^{\varepsilon_{k_r}})^{-1}\big)$$

$$\approx \tau_m\big(w_1(g_i)R_{j_1}w_1^{-1}(g_i) \dots w_p(g_i)R_{j_p}w_p^{-1}(g_i)\big)$$

$$\approx \tau_m\big(w_1(g_i)R_{j_1}w_1^{-1}(g_i)\big) \dots \tau_m\big(w_p(g_i)R_{j_p}w_p^{-1}(g_i)\big) \text{ by property (2).}$$

Let $\bar{1}w(g_i) = V_{1,w}(h).\bar{\lambda}$ where $\bar{\lambda}$ is a coset representative. Then $\bar{\lambda}w^{-1}(g_i) = V_{1,w}^{-1}(h).\bar{1}$, from (24).

Now, $\bar{\lambda}R_j(g_i) = V_{\lambda,R_j}(h).\bar{\lambda}$ so, $\bar{1}w(g_i)R_j w^{-1}(g_i) = V_{1,w}(h).\,\bar{\lambda}R_j w^{-1}(g_i)$

$$= V_{1,w}(h)\,V_{\lambda,R_j}(h).\,\bar{\lambda}\,w^{-1}(g_i)$$

$$= V_{1,w}(h)\,V_{\lambda,R_j}(h)\,V_{1,w}^{-1}(h).\bar{1}\,.$$

Thus,

$$\tau_m\big(w(g_i)R_j w^{-1}(g_i)\big) = V_{1,w}(h)\,V_{\lambda,R_j}(h)\,V_{1,w}^{-1}(h) \qquad (26)$$

for any word $w(g_i)$ and any relator R_j, $1 \le j \le t$.

By condition (2),

$$\tau_m\left(U(g_i)(J_{k_1}(g_i)^{\varepsilon_{k_1}} J_{k_2}(g_i)^{\varepsilon_{k_2}} \ldots J_{k_r}(g_i)^{\varepsilon_{k_r}})^{-1}\right)$$

$$\approx \tau_m(U(g_i))\,\tau_m(J_{k_r}(g_i)^{-\varepsilon_{k_r}}) \ldots\ldots \tau_m(J_{k_2}(g_i)^{-\varepsilon_{k_2}})\,\tau_m(J_{k_1}(g_i)^{-\varepsilon_{k_1}})$$

$$= V_{1,U}(h)\, V_{1,J_{k_r}^{-\varepsilon_{k_r}}}(h) \ldots\ldots V_{1,J_{k_2}^{-\varepsilon_{k_2}}}(h)\, V_{1,J_{k_1}^{-\varepsilon_{k_1}}}(h)\,.$$

$$\therefore\ V_{1,U}(h)\, V_{1,J_{k_r}^{-\varepsilon_{k_r}}}(h) \ldots\ldots V_{1,J_{k_2}^{-\varepsilon_{k_2}}}(h)\, V_{1,J_{k_1}^{-\varepsilon_{k_1}}}(h)$$

$$= V_{1,w_1}(h)\, V_{\lambda_1,R_{j_1}}(h)\, V_{1,w_1}^{-1} \ldots\ldots (h) V_{1,w_p}(h)\, V_{\lambda_p,R_{j_p}}(h)\, V_{1,w_p}^{-1}(h)\,. \quad (27)$$

Now, $\bar{\lambda} R_j(g_i) = \bar{\lambda}\,1 = 1.\bar{\lambda}$ for any coset representative, $\bar{\lambda}$.

Therefore, $V_{\lambda,R_j}(h)=1\ \forall\ \bar{\lambda}$, $1 \le \lambda \le |\mathbf{G}:\mathbf{H}|$, and $\forall\ j$, $1 \le j \le t$.

Thus $V_{\lambda,R_j}(h)$ is a relator in **H**, so any conjugate of it is also a relator. Thus the right hand side of (27) is equal to 1. Hence,

$$V_{1,U}(h) = V_{1,J_{k_1}^{-\varepsilon_{k_1}}}^{-1}(h)\, V_{1,J_{k_2}^{-\varepsilon_{k_2}}}^{-1}(h) \ldots\ldots V_{1,J_{k_r}^{-\varepsilon_{k_r}}}^{-1}(h)\,. \quad (28)$$

Now, $\bar{1} J_k(g_i)^{\varepsilon_k} = V_{1,J_k^{\varepsilon_k}}(h).\bar{1}$

$$\Rightarrow \bar{1}\, J_k(g_i)^{-\varepsilon_k} = V_{1,J_k^{\varepsilon_k}}^{-1}(h).\bar{1}\,.$$

$$\therefore\ V_{1,J_k^{-\varepsilon_k}}(h) = V_{1,J_k}(h)^{-\varepsilon_k}. \quad (29)$$

Hence, using (29), (28) becomes

$$V_{1,U}(h) = V_{1,J_{k_1}}(h)^{\varepsilon_{k_1}} \ldots\ldots V_{1,J_{k_r}}(h)^{\varepsilon_{k_r}}\,. \quad (30)$$

From the construction of the coset tables,

$$\bar{1} J_k(g_i) = h_k.\bar{1} \ \text{ so } V_{1,J_k}(h) = h_k \text{ holds } \forall\ k,\ 1 \le k \le s.$$

Therefore, the right hand side of (30) is the same element as :

$$h_{k_1}^{\varepsilon_{k_1}} h_{k_2}^{\varepsilon_{k_2}} \ldots h_{k_r}^{\varepsilon_{k_r}}\,,$$

i.e. $\tau_m(U(g_i))$ is the same element as $h_{k_1}^{\varepsilon_{k_1}} h_{k_2}^{\varepsilon_{k_2}} \ldots h_{k_r}^{\varepsilon_{k_r}}$ which by definition is $J_{k_1}(g_i)^{\varepsilon_{k_1}} J_{k_2}(g_i)^{\varepsilon_{k_2}} \ldots J_{k_r}(g_i)^{\varepsilon_{k_r}} = U(g_i)$.

Hence τ_m is a rewriting process.

Using Theorem 3 on page 14 , the presentation for **H** is

$$\langle h_1,\ldots, h_s \mid h_k = \tau_m(J_k(g_i)),\ \tau_m(w R_j w^{-1}) = 1;\ 1 \le k \le s,\ 1 \le j \le t \rangle$$

where $R_j(g_i)$ is a defining relator in (4) and w is any word in the elements of $\mathbf{X}^{\pm 1}$.

The first set of relations, $h_k = \tau_m(J_k(g_i))$, are just those obtained from the subgroup generator tables in the modified algorithm as described on page 31.

$\bar{1}J_k(g_i) = h_k \cdot \bar{1}$ from the subgroup table so $V_{1,J_k}(h).\bar{1} = h_k.\bar{1}$. Hence, $V_{1,J_k}(h) = h_k$ is a relation i.e. $\tau_m(J_k(g_i)) = h_k$.

Tying up the final set of relations above with those in (22) on page 31 is slightly more difficult.

By (26) $\tau_m\big(w(g_i)R_j w^{-1}(g_i)\big) = V_{1,w}(h)\, V_{\lambda,R_j}(h)\, V^{-1}_{1,w}(h)$ where $\bar{\lambda}$ is some coset representative s.t. $\bar{1}w(g_i) = V_{1,w}(h).\bar{\lambda}$.

Now, any conjugate of a relation is also a relation, so $w(g_i)R_j w^{-1}(g_i) = 1$. Thus by the definition of a rewriting process, $\tau_m\big(w(g_i)R_j w^{-1}(g_i)\big) = \tau_m(1) = 1$,

$$\text{i.e. } V_{1,w}(h)\, V_{\lambda,R_j}(h)\, V^{-1}_{1,w}(h) = 1.$$

Now, $V_{1,w}(h)\, V_{\lambda,R_j}(h)\, V^{-1}_{1,w}(h) = 1 \Leftrightarrow V_{\lambda,R_j}(h) = 1$.

Thus, $V_{\lambda,R_j}(h) = 1 \Rightarrow V_{1,w^*}(h)\, V_{\lambda,R_j}(h)\, V^{-1}_{1,w^*}(h) = 1$ for all words $w^*(g_i)$ s.t. $\bar{1}w^*(g_i) = V_{1,w^*}(h).\bar{\lambda}$.

Conversely, for each $w(g_i) \in \mathbf{G}$ $\exists$ a coset representative $\bar{\lambda}$, $1 \le \lambda \le |\mathbf{G}:\mathbf{H}|$ s.t. $\bar{1}w(g_i) = V_{1,w}(h).\bar{\lambda}$.

$$\therefore V_{\lambda,R_j}(h) = 1 \Rightarrow V_{1,w}(h)\, V_{\lambda,R_j}(h)\, V^{-1}_{1,w}(h) = 1.$$

Hence the second set of relations can be obtained from the conjugation by elements of **G** of the set of relations $\mathbf{R}' = \{ V_{\lambda,R_j}(h) = 1;\ 1 \le \lambda \le |\mathbf{G}:\mathbf{H}|,\ 1 \le j \le t \}$. $\mathbf{R}'$ is just the set of relations (22) obtained from the modified algorithm. Therefore, the modified Todd-Coxeter process as described in Section 1.5 is a rewriting process which allows us to find a presentation for a subgroup **H** of **G**.

Using the Reidemeister-Schreier and modified Todd-Coxeter processes two different presentations can be found for a subgroup **H** in a group **G**. Since **H** is the same group in each case, it must be possible to obtain one presentation from the other via a finite sequence of Tietze transformations (see Section 1.3). In a Reidemeister-Schreier process the coset representatives always form a Schreier system. However, in the modified Todd-Coxeter process, although the coset representatives are defined in a "Schreier way", the resulting set of coset representatives may not form a Schreier system if coincidences have occurred (i.e. there may not be enough empty words in the final augmented coset table).

If we use a Reidemeister rewriting process, relaxing the condition that the transversal must be Schreier, we end up with the following presentation

$$\langle s_{K,g_i} \mid s_{K,g_i} = \tau(K g_i \overline{K g_i}^{-1}), \tau(K R_j K^{-1}) = 1 \rangle \qquad (31)$$

as proved in Section 1.4. Now, each of the generators $s_{K,g_i} \in \mathbf{H}$ can be written in terms of the given subgroup generators h_k i.e. $s_{K,g_i} = w(h_k)$. Similarly, each generator h_k can be written as a word in the generators s_{K,g_i} i.e. $h_k = w^*(s_{K,g_i})$. Under a finite number of Tietze transformations presentation (31) then becomes equivalent to

$$\langle s_{K,g_i}, h_k \mid s_{K,g_i} = \tau(K g_i \overline{K g_i}^{-1}), \tau(K R_j K^{-1}) = 1, s_{K,g_i} = w(h_k), h_k = w^*(s_{K,g_i}) \rangle.$$

..... (32)

To find a presentation solely on the generators h_k we now want to eliminate all of the Schreier generators s_{K,g_i}. We do this by substituting the equations $s_{K,g_i} = w(h_k)$ into each of the other relations in (32).

Since our choice of transversal for the Reidemeister rewriting process is completely free we can choose the same transversal as that used for the modified Todd-Coxeter algorithm. Thus, in the Reidemeister rewriting process, $s_{K,g_i} = K g_i \overline{K g_i}^{-1}$, and in the modified Todd-Coxeter process, $K g_i = v_{K,g_i}(h) \overline{K g_i}$ from the augmented coset table

i.e. $v_{K,g_i}(h) = K g_i \overline{K g_i}^{-1} = s_{K,g_i}$.

This then simplifies (32), since $s_{K,g_i} = w(h_k)$ becomes

$$s_{K,g_i} = v_{K,g_i}(h) \qquad (33)$$

[i.e. the h-words in the augmented coset table are just the Schreier generators].

$\underline{\tau(K R_j K^{-1}) = 1}$

Let $K = g_{K_1} g_{K_2} \cdots g_{K_r}$ and $R_j = g_{j_1} g_{j_2} \cdots g_{j_p}$. Then,

$$\tau(K R_j K^{-1}) = \tau((g_{K_1} g_{K_2} \cdots g_{K_r})(g_{j_1} g_{j_2} \cdots g_{j_p})(g_{K_r}^{-1} \cdots g_{K_2}^{-1} g_{K_1}^{-1})) = 1$$

$$\Rightarrow \quad (s_{1, g_{K_1}} \cdots s_{\overline{g_{K_1} g_{K_2} \cdots g_{K_{r-1}}}, g_{K_r}})(s_{K, g_{j_1}} \cdots s_{\overline{K g_{j_1} g_{j_2} \cdots g_{j_{p-1}}}, g_{j_p}})$$
$$(s^{-1}_{\overline{g_{K_1} g_{K_2} \cdots g_{K_{r-1}}}, g_{K_r}} \cdots s^{-1}_{1, g_{K_1}}) = 1$$

$$\Rightarrow \quad (v_{1, g_{K_1}} \cdots v_{\overline{g_{K_1} g_{K_2} \cdots g_{K_{t-1}}}, g_{K_t}})(v_{K, g_{j_1}} \cdots v_{\overline{K g_{j_1} g_{j_2} \cdots g_{j_{p-1}}}, g_{j_p}})$$
$$(v^{-1}_{\overline{g_{K_1} g_{K_2} \cdots g_{K_{r-1}}}, g_{K_r}} \cdots v^{-1}_{1, g_{K_1}}) = 1 \text{ from (33)},$$

i.e. $\tau_m(K(g_i) g_{j_1} g_{j_2} \cdots g_{j_p} K(g_i)^{-1}) = 1$,

i.e. $\tau_m(K R_j K^{-1}) = 1$,

the second set of relations in the presentation on page 40. Thus, when $s_{K, g_i} = v_{K, g_i}(h)$ is substituted into the second set of relations in (32) we get the relations obtained from the relation tables in the modified algorithm.

$\underline{h_k = w^*(s_{K, g_i})}$

Let $J_k(g_i) = g_{k_1} g_{k_2} \cdots g_{k_p}$. Now, $h_k = J_k(g_i)$ and $\tau(J_k(g_i)) = s_{1, g_{k_1}} \cdots s_{\overline{g_{k_1} g_{k_2} \cdots g_{k_{p-1}}}, g_{k_p}}$, so since τ is a rewriting process, h_k is the same element as $s_{1, g_{k_1}} \cdots s_{\overline{g_{k_1} g_{k_2} \cdots g_{k_{p-1}}}, g_{k_p}}$. Therefore, $h_k = \tau(J_k(g_i))$ are a suitable set of relations for the fourth set in (32). Substituting $s_{K, g_i} = v_{K, g_i}(h)$ into these we obtain

$$h_k = v_{1, g_{k_1}} \cdots v_{\overline{g_{k_1} g_{k_2} \cdots g_{k_{p-1}}}, g_{k_p}}$$
$$= \tau_m(J_k(g_i)) .$$

These are exactly those relations obtained from the subgroup tables in the modified algorithm. Hence we have shown that the presentation on page 40 can be derived from (31).

However, we are left with the first set of relations in (32). When we substitute $s_{K, g_i} = v_{K, g_i}(h)$ into them we must get a set of relations which are consequences of the others. This is easy to show in the case that the transversal is Schreier.

In this case we can choose the same Schreier transversal for the Reidemeister rewriting process.

Then, $s_{K,g_i} = \tau(K g_i \overline{K g_i}^{-1}) = K g_i \overline{K g_i}^{-1}$ from the proof of Theorem 6

i.e. $s_{K,g_i} = s_{K,g_i}$.

Thus, this set of relations reduces to $v_{K,g_i} = v_{K,g_i}$ – a set of trivial relations.

If the transversal is not Schreier, I do not see an easy way of proving this.

This method used here, of obtaining a presentation of the form (32) and eliminating the Schreier generators, is basically that described in Mendelsohn [29]. His Schreier generators are already written in terms of the original subgroup generators h_k. He has to find a way of writing the given subgroup generators of **H** in terms of the Schreier generators. He does this by first carrying out a second coset enumeration using the Schreier generators as the defining generators of **H**. Then, since words in the s_{K,g_i} appear in the augmented coset table, by tracing through $J_k(g_i) = h_k$ with coset 1, concatenating the s-words, an expression for h_k in terms of the Schreier generators is found. Now the generators s_{K,g_i} can be eliminated leaving a presentation on the given subgroup generators, h_k.

Chapter 2

Implementations of Computer Programs

In this chapter I will give a brief history of the use of computers to carry out coset enumeration and describe various computer programs which implement the algorithms described in Chapter 1.

§2.1 The Use of Computers for Coset Enumeration

Over the past three decades increasing use has been made of computers to carry out research in abstract algebra. Although computers have been used in such diverse branches of the subject as Combinatorics, Number Theory and Semigroups, by far the majority of the applications have been in the field of Group Theory.

The first known reference to computing with groups is a proposal to investigate p-groups made by M.H.A. Newman in his article "The Influence of Automatic Computers on Mathematical Methods" published in the *Proceedings of the Inaugural Conference of the Manchester University Computer* in 1951. However, it is thought that his idea was never actually implemented.

Very early on it was realised that a Todd-Coxeter method could be easily adapted for automatic execution on a computer. Just two years after Newman's article appeared, C.B. Haselgrove wrote a program to perform coset enumeration on EDSAC1 at Cambridge. In common with all later implementations of the algorithm, apart from the initial data, Haselgrove stored only the coset table and not the subgroup generator or relation tables. Each row of the latter two sets of tables is then reconstructed as required, using the entries in the coset table, in order to find new deductions or coincidences. This practice is followed because storage space rather than computing time tends to be the limiting factor on the size of enumerations that can be tackled. Haselgrove stored the multiplication table for the cosets in such a way that each register held the effect of a generator and also its inverse, on a coset. Thus, the number of registers required for a coset table was the product of the number of cosets and the number of generators. As the store only had 512 registers, and these also had to hold the program and other working space, no large examples could be tackled. Basically, the program only proved that it was possible to run the Todd-Coxeter coset enumeration algorithm on a digital computer.

When EDSAC2 was installed at Cambridge in 1958, P. Maddison and J. Leech wrote new programs based on Haselgrove's work. Leech used only one column of the coset table for an involutory generator, and each storage register held four table entries instead of two. He wrote two versions of the program - one allowing at most four columns per coset, the other eight columns. After storing the program and allowing space for other working, this left room for a maximum of 800 or 400 cosets respectively - a vast improvement on the capabilities of Haselgrove's program.

Haselgrove then wrote another program for the Mercury at Manchester in 1960. An interesting feature of this implementation was that it used a two-level store of magnetic cores and magnetic drum. The coset table was stored consecutively on the drum, at any instant three sections of it also being held in the core store. Certain sections in the core store were "reserved" when it was known that they would be wanted again soon, then when a section not in the core store was required, an unreserved section was returned to the drum to make way for the new one. The sections containing the current coset and the next to be defined were reserved in normal working, and the sections containing two equivalent cosets reserved when processing a coincidence. This system reduced the number of transfers required in the course of the enumeration.

The next known program was written by A. Sinkov for a 704 in 1962, then yet another implementation was described by H.F. Trotter in 1964 [35]. He claimed that he could define a maximum of 29000/(n+2) cosets, where n is the number of group generators. This was implemented on an IBM 7090/94. Trotter also gave a proof of the algorithm in terms of arrays of integers and chains of coset number-generator pairs.

All of these programs, although different, essentially use a common method of defining cosets and processing the relations. They accept coincidences as being inevitable and do not try to minimize their occurrence. Instead, the enumeration procedure is simplified so as to shorten the program as much as possible and leave a maximum amount of storage space for the coset table. Each coset is taken in turn and applied to each relation in turn. If α_0 is the current coset and $g_{i_1} g_{i_2} \cdots g_{i_k} = 1$ the current relation, the successive entries $\alpha_1 = \alpha_0 g_{i_1}$, $\alpha_2 = \alpha_1 g_{i_2}$, $\ldots$, $\alpha_k = \alpha_{k-1} g_{i_k}$ are extracted from the table. If any of these entries has not been defined, a new coset $\lambda+1$ (where λ is the last previously defined coset) is defined as $\lambda+1 = \alpha_j g_{i_{j+1}}$ and

this, and its inverse equation $(\lambda+1)\, g_{i_{j+1}}^{-1} = \alpha_j$, entered in the table. Further new cosets are defined in a similar manner until the end of the relation is reached. At this point, α_0 is compared to α_k, and if they are different the coincidence procedure is called in.

When the current coset has been applied to all the relators (and the subgroup generators if it happens to be 1) the next coset in numerical order is taken as the current coset and applied to all of the relators in a similar way. This process continues until the last coset in the table has been applied to the last relator, or until the store is exhausted.

This method has been called the HLT method, a term coined by John Cannon [15] and named after its three main developers - Haselgrove, Leech and Trotter.

Another method developed in parallel to the HLT method. In this method, after each definition is made, a scan of all relators is carried out to ensure that no deduction is missed. This scan is started at all significantly different parts of each relator into which the definition fits† and continued both to the right and left of the definition, cyclically permuting the relator as necessary , until either a gap is found or the relator closes. This method was first programmed by Bandler in 1956. No attempt was made to deal with coincidences in the program. If one occurred, the multiplication table was printed out together with details of the coincidence and it was dealt with by hand, the new information being fed in to restart the enumeration. Various different rules were tried for choosing the order in which cosets were defined in the hope of avoiding coincidences. These included filling in the earliest blank in the multiplication table and the earliest gap in the column after the one belonging to the previous definition, if this was not full.

H. Felsch (1961) wrote a similar program for the Zuse 22 machine at Kiel. His method was almost the same as Bandler's, the main difference being that the machine processed the coincidences. The method for dealing with the coincidences is much more complicated than that for the HLT method, as both the deduced coincidences and the deduced entries in the multiplication table have to be stored for subsequent working. The latter may be changed by coincidences so the list has to be continually updated.

---

† By this we mean that if a relator expresses the period of some word, w^n say, then we need only look at each occurrence of our chosen generator or its inverse in the word w, and not in the rest of the relator.

This second variation of Todd-Coxeter coset enumeration is usually known as the "Felsch" method, since Felsch implemented the first complete program.

Both methods have advantages and disadvantages in the extent to which the different strategies affect the time and space needed to complete the enumeration. In the HLT method a large number of redundant cosets are defined and subsequently eliminated when coincidences are found. This wastes some time, but, more importantly, may completely fill up the storage space, especially if the relations are lengthy. The second method, since every possible consequence of each definition is methodically searched for, tends to be slower, but is very economical on space, often completing with the definition of few (or no) redundant cosets.

A third method has been developed more recently which combines the other two. This is the "Lookahead" method. In this, cosets are defined as in HLT until either a pre-defined limit on the number of active cosets has been reached, or the storage space is exhausted. At this point the Felsch method comes into action scanning each relation with each active coset. No new definitions are made but all deductions and coincidences found are processed and the new information entered in the coset table, [i.e. the computer looks ahead to all rows in the relation tables which have not yet been processed in the first phase, to see if they close, yielding new information]. When all possible information has been extracted from the relations, the defining HLT phase is re-entered. Thus, we alternate between the two methods until the enumeration completes or the lookahead does not release any space.

A type of Lookahead was used by Leech in 1959 [24] for one stubborn example, but this variation did not really begin to develop until M. J. T. Guy wrote a Lookahead program for the ATLAS at Cambridge in 1967.

Obviously, the characteristics of the example chosen affect the running of the different methods. The efficiency of all implementations can be substantially changed by such seemingly trivial modifications as altering the order of the relators, cyclically permuting individual relators or adding redundant relators or subgroup generators. This was shown in 1973 by J.J. Cannon, L.A. Domino, G. Havas and J.M. Watson [15]. They studied the behaviour of the Todd-Coxeter programs in a wide variety of situations and also compared the performances of the three methods for a selection of different examples. They concluded that the Lookahead algorithm gave the best all-round results. It seemed to be less-affected by examples which were found 'difficult'

by HLT or Felsch. In general it was as fast, if not faster, than the other two algorithms and did not use much more space than Felsch.

I will now go on to describe some implementations of these different methods and a version of the modified Todd-Coxeter process based on HLT. (In each case I will describe the implementation running on the VAX 11/785 in St Andrews.)

§2.2 Implementation of the Todd-Coxeter Process - TC1

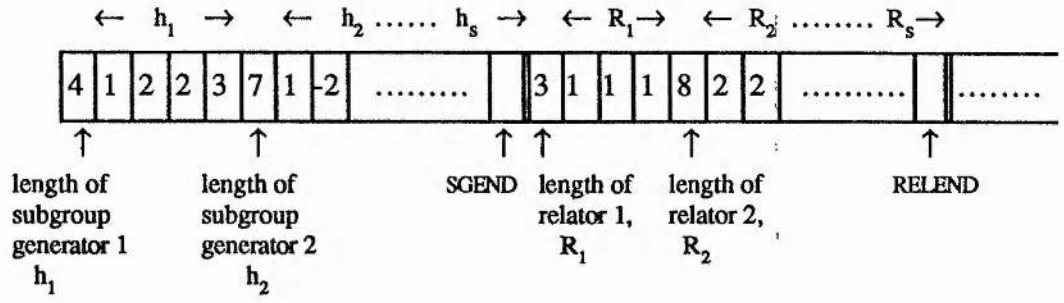
The first program I will describe, TC1, is the version of HLT with Lookahead on to which the modified Todd-Coxeter algorithm described in Section 2.3 has been "grafted". It is very similar to that described by Cannon et al [15]. It is written in FORTRAN 66 and uses a linked list to keep track of which cosets have been defined and which have been subsequently eliminated due to coincidences. In early programs this "freed space" due to coincidences was collected together by moving all the active rows to one end of the store and systematically renumbering these cosets. This was done after each session of coincidence processing before returning to the defining phase, and was very time consuming, especially if low numbered cosets disappeared.

The most critical part of implementing any Todd-Coxeter algorithm is selecting the data structures to be used. In this version one large array called SPACE is set up. This stores the subgroup generators, the group relators, the coset table and the active coset list. The size of this array can be altered to suit the problem and the capabilities of the machine used.

The data can either be read from a file or entered interactively from the terminal. Before putting in the subgroup generators and group relators it is possible to enter a value to NMAXX. This restricts the number of cosets defined at any one time to NMAXX i.e. allows the table to grow to a maximum of NMAXX rows before a lookahead takes place. If no value is given, the maximum possible number of rows is defined to be the largest possible integer with respect to the storage space available inside SPACE. The subgroup generators (if any) are then read in, followed by the group relators. The program can cope with up to 26 group generators, input as A-Z. The subgroup generators are input on separate lines, the input ending with "!" on a line of its own. Each generator is entered as a string of characters made up of upper case letters, numbers, minus signs (used to denote inverses), square and round

brackets. For example $AB^{-4}(AB^3)^2$ is entered as A-B4(AB3)2. An "&" is used as a continuation character if the string is longer than 79 characters. The group relators are entered in the same format, also ending with an exclamation mark.

The subroutine GENREL is used to process these subgroup generators and group relators and translate them into strings of numbers from 1 to 26 or their negatives (denoting the inverse of a generator). The string of characters is processed from left to right, expanding all brackets and writing out the word in full without exponents. The characters are converted into numbers using the subroutine CONV. The finished generator or relator is then written into the array, preceded by the length of the string. A variable called SGEND marks the end of the subgroup generators and the start of the group relators, and RELEND the end of the group relators.



While this initial data is being entered a 1-dimensional array called GENCOL keeps track of which of the possible 26 group generators are being used. GENCOL(I) is initially $\emptyset$ but is changed to 1 if the Ith letter of the alphabet is used as a generator in the presentation, -1 if it is an involutory generator. A variable, NCOL, is used to denote the number of columns in the coset table. It is initially set at 2, these two columns being reserved to hold the coincidence queue and active coset lists. After the relators are entered, the array GENCOL is then checked. If GENCOL(I)=1, $1 \leq I \leq 26$, NCOL is increased by two and if GENCOL(I)=-1, NCOL is increased by one. Note that involutory generator relators are not stored in the list of relators since any new information to be gained from them will be picked up anyway from the coset table.

Two new 1-dimensional arrays are now introduced and the role of GENCOL changed :-

INVCOL is set up so that INVCOL(I) holds the column number for the inverse of the generator or its inverse corresponding to column I.

GENCOL is set up so that GENCOL(I) gives the column number for generator I.

INGCOL is set up so that INGCOL(I) is the column number for the inverse of generator I.

These three arrays do not form part of the large array SPACE. The subgroup generators and group relators are translated from strings in terms of the generators and their inverses into strings in terms of their associated column numbers using GENCOL and INGCOL. Note that now that we know the number of columns in the coset table, NCOL, we can calculate the maximum number of cosets that will fit into the remainder of SPACE. As NMAX, the maximum number of rows allowed in the coset table, we take the minimum of this calculated value and the value of NMAXX.

We are then ready to do the enumeration, which is carried out in the subroutine ENUM. This is called with the parameter SPACE(RELEND+1) which becomes the first entry of the 2-dimensional array SPACE in ENUM. The original large array SPACE is now called SGREL. [This causes some confusion on an initial examination of the program !] This new array SPACE is basically the coset table with NCOL columns and NMAX rows. For an active coset, SPACE(I, 1) is a pointer to the last previously-defined active coset, SPACE(I, 2) a pointer to the next active coset. SPACE(I, J), $J = 3, \ldots,$ NCOL, indicates the action of the associated generator (or inverse) of column J on coset I. It will contain the resulting coset number, if known, otherwise $\emptyset$. Coset 1 is the subgroup itself, so already exists. Therefore, row 1 of the coset table can be initialized straightaway. SPACE(I, 2) is used to link up the free space throughout the table. Initially, SPACE(I, 2) is set equal to I+1, $I = 2, \ldots,$ NMAX-1 and SPACE(2, NMAX) set to $\emptyset$. A variable FREEL is then initialized as 2, the first free coset number .

There are two parts to the enumeration - processing the subgroup generators and processing the relators. However, since the same methods are used I will describe them together.

Starting with the first subgroup generator, we scan forwards through the word (i.e. from left to right) with coset 1 until either the end is reached or we find a zero in the coset table. If the former happens we note the resulting coincidence (if it is non-trivial) - see later description of coincidence handling. If a zero is reached, another scan is started in the backward direction i.e. starting at the end of the relator and working from right to left. When the scan can proceed no further in this direction, new cosets are defined to fill any remaining gap. Therefore, defining is only done on the backward scan of a subgroup generator or relator. When finally there is no gap,

i.e. the two scans completely fill the row of the relation or subgroup generator, we get a deduction. However, at this point we must make sure that the inverse deduction in the forward direction is consistent with the coset table. If it is not, we have a coincidence.

Thus, when a scan of a subgroup generator or group relator closes we get either a deduction, no new information or a coincidence (see page 4). If it is a deduction the new information is entered in the coset table. If we have a coincidence, the coincidence processing subroutine COINC is called - I will describe this in detail later.

At any instant the next coset to be defined is FREEL, and once it has been defined, the value of FREEL is updated to SPACE(FREEL, 2), the next available coset in the list of free space. If this turns out to be $\emptyset$ we know that no more cosets can be defined (i.e. we have defined NMAX cosets) and we have to enter the lookahead phase. When a new coset I is defined, SPACE(I, 1) is made equal to the last active coset LASTDF and SPACE(LASTDF, 2) is changed to I, linking the new coset into the active coset list.

When the subgroup generators have been scanned in turn with coset 1, and new cosets defined where needed until closure, 1 is applied in a similar manner to the group relators. Any coincidences found are processed as they are discovered. Then the first row of the coset table is examined. If there are any gaps, new cosets are defined to fill these, and the appropriate rows of SPACE initialized. Now coset 1 is finished with and coset 2 is taken to be the new current coset KN. The same procedure is followed with this coset (omitting the subgroup generators of course). This process continues until either the algorithm terminates (i.e. each relator is closed under scans with all active cosets and there are no gaps in the coset table) or we run out of space (i.e. we already have NMAX active cosets and need to define more). If the latter happens we enter a lookahead phase. This applies all cosets larger than LCLOSE† to each relation in turn to try to find new deductions and coincidences to close the coset table or to free space in it.

If a coset with a higher number than some unclosed coset is found to be closed, it is linked around in the active coset list and relinked just after LCLOSE. The

---

† LCLOSE is the last coset under which all relators have closed and which has no gaps in its row in the coset table.

value of LCLOSE is then updated to this new coset number. This saves time in future lookaheads and in the defining phase as this coset does not have to be used for unnecessary scans. No new cosets are defined until all the relators have been processed with all "unclosed cosets" and all resulting coincidences dealt with.

[Note that one lookahead does not extract all possible new information from the relators. Deductions obtained near the end of the lookahead may help earlier unclosed rows in the relation tables to now close, giving more deductions. However, this information will not be found until that row is processed in the defining phase or we run out of space again and enter a new lookahead. For this reason, some programs make more than one "pass" during the lookahead phase.]

The defining phase is then re-entered with KN equal to the last closed coset LCLOSE (which may have been updated during the lookahead phase). The enumeration alternates between the defining and lookahead phases until the enumeration completes or no space is recovered during a lookahead. If the latter occurs, and there is still room left in SGREL, the user is asked to input a larger value of NMAXX. If a smaller one is given by mistake, the program stops, otherwise the enumeration is restarted from scratch.

Before each lookahead commences, the number of active cosets, NALIVE, is written out, along with the maximum number of cosets active at any one time, MAXCOS, and the total number of cosets defined, TOTCOS. Then, after the lookahead is complete, a message giving the number of cosets remaining is output.

When the enumeration terminates, the index of the subgroup (the current value of NALIVE) and the values of MAXCOS and TOTCOS are written to the screen.

Coincidence Handling

Coincidence handling is the most complicated part of any Todd-Coxeter program. In this case the coincidences are queued in the first column of SPACE and marked with a negative flag in the second. QHEAD is a variable pointing to the top of the list of coincidences, and QTAIL to the bottom. Any new coincidences resulting from the processing of the coincidences in the queue are added to the bottom and QTAIL altered accordingly. The larger of the two cosets is always marked as being coincident to the other one, not vice versa.

E.g.

	1	2	...
1	0	3	
2	0	-1	
3	1	4	
4	3	6	
5	2	-3	
6	4	8	
7	5	-4	
8	6	9	
⋮	⋮	⋮	

In this case, QHEAD=7, QTAIL=2 and the coincidences are processed in the order

$7 \equiv 4 \quad 5 \equiv 3 \quad 2 \equiv 1$

If a new coincidence is found, $\alpha \equiv \beta$ say, then SPACE(α, 2) is examined If this value is already marked as being coincident to another coset, γ say, then SPACE(γ, 2) is inspected to see if it is coincident to yet another coset. In this way we work our way down the chain to find the lowest numbered coset that α is coincident to, α' say. Then we repeat the process with β to find β' and queue the resulting coincidence $\alpha' \equiv \beta'$. We do this by choosing $\beta''=\max(\alpha', \beta')$ and $\alpha''=\min(\alpha', \beta')$, linking around β'' in the active coset list and then setting SPACE(β'', 2)=$-\alpha''$. Obviously, if $\alpha''=\beta''$ we have a trivial coincidence and nothing is entered in the coset table. Thus, this method of finding the smallest coset in the chain each time ensures that little redundant information is entered in the coincidence queue. As soon as there is one coincidence in the queue, no more scanning is done, in either the defining or lookahead phases, until the list is empty again. In the program the larger of the two coincident cosets is called KB, the smaller KA.

We now process the first coincidence in the queue. The two rows of the coset table to be compared are obtained using the following code :

```
KB = QHEAD
QHEAD = SPACE(KB, 1)
KA = −SPACE(KB, 2)
```

Then KB is linked onto the beginning of the list of free space :

$$\text{SPACE}(KB, 1) = \emptyset$$
$$\text{SPACE}(KB, 2) = \text{FREEL}$$
$$\text{FREEL} = KB$$

i.e. the next free coset number is KB.

Now we want to compare corresponding columns in each of these two rows, queueing any more coincidences that become apparent. This is done in the following manner :

We use variables JB and JA to denote the entries in the Ith column of KB and KA respectively.

$$JB = \text{SPACE}(KB, I)$$
$$JA = \text{SPACE}(KA, I)$$

(1) We can now have one of three possibilities for JB.

(a) $JB=\emptyset$. In this case we do nothing and go on to the next value of I.

(b) JB=KB. Here we change the value of JB to KA and look at JA as in (2) below.

(c) $JB\neq\emptyset$, JB≠KB. Here we delete SPACE(JB, INVCOL(I)) (which is KB). We do this rather than replace it by KA to avoid two occurrences of KA in the same column.

(2) Now we look at JA. Again there are three possibilities.

(a) $JA=\emptyset$. Here we get a deduction so can fill in SPACE(KA, I)=JB.

(b) JA=KB. In this case we change JA to KA and alter SPACE(KA, I) TO KA. We then queue the coincidence JA≡JB if it is not trivial or already on the queue.

(c) $JA\neq\emptyset$, JA≠KB. Here we queue JA≡JB as in (b) above.

Now we can fill in the inverse entry SPACE(SPACE(KA, I), INVCOL(I))=KA if it is currently undefined.

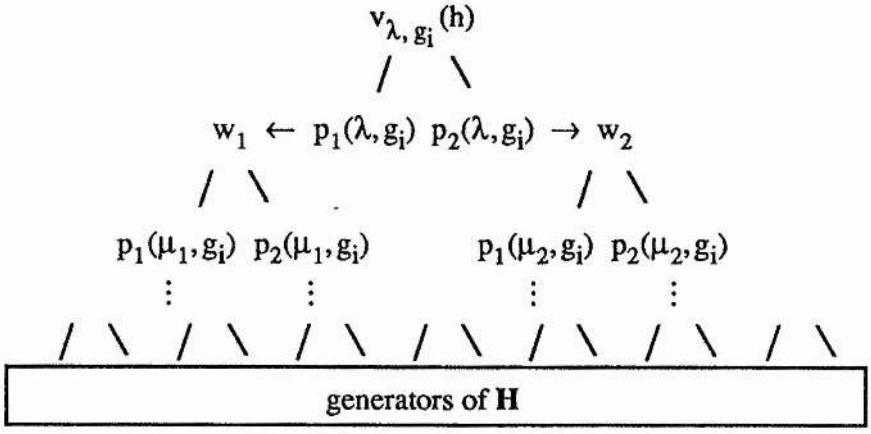
This is done for each value of I between 3 and NCOL, then the next coincidence KB≡KA is extracted from the queue to be similarly processed.

§2.3 Implementation of the Modified Todd-Coxeter Algorithm

One major problem with implementing this method is that when we concatenate the h-words $v_{\lambda, g_i}(h)$ (see page 29) each time we obtain a deduction or process a coincidence these new words may become very long. This greatly increases the storage requirements of the algorithm and, since each word is of variable length, it is impractical to store these in full inside the augmented coset table as in the hand method.

One solution to this problem is described in [1]. Each h-word is constructed recursively as the algorithm progresses in the form $V_{\lambda, w(g_i)}(h) = w_1^{\varepsilon_1} w_2^{\varepsilon_2}$; $\varepsilon_1, \varepsilon_2 = \pm 1$ where w_1 and w_2 are two previously defined words in the subgroup generators h_k. Instead of storing the final deduction word, $v_{\alpha, g_i}(h)$ in the coset table, a pair of pointers $p_1(\alpha, g_i)\ p_2(\alpha, g_i)$ is stored instead, pointing to the words w_1 and w_2 respectively. (If ε_1 or $\varepsilon_2 = -1$, we store the negative of the appropriate pointer.) In turn, the words w_1, w_2 may also be stored as the product of two pointers and so on. Thus, as well as storing the augmented coset table, we also need to store a binary tree, containing the information which allows us to construct the V(h) from sequences of pointers $p_j(\lambda, g_i)^{\varepsilon}$, $j = 1, 2$; $\varepsilon = \pm 1$. Although this method saves considerably on storage space, we have to perform a lot of tree searching in order to construct the subgroup relations in terms of the original subgroup generators at the end of the enumeration. However there is a possible shortcut we can take here. When we collect these relations we can treat the pointers, $p_j(\lambda, g_i)^{\varepsilon}$, as generators for **H** and, postponing any decoding of pointers at this stage, obtain a presentation on $h_1, h_2, \ldots, h_s, p_j(\lambda, g_i)^{\varepsilon}$; $j = 1, 2$; $\varepsilon = \pm 1$; $\lambda = 1, \ldots, |\mathbf{G} : \mathbf{H}|$; $i = 1, \ldots, n$. These relations are considerably shorter than those which would have been found in terms of the h_k alone and can often be greatly simplified using Tietze transformations. These transformations have a good chance of eliminating some or all of the $p_j(\lambda, g_i)^{\varepsilon}$, doing away with the need to translate these back into words in the original subgroup generators, h_k. If some $p_j(\lambda, g_i)^{\varepsilon}$ cannot be eliminated, we read from the tree that

$p_j(\lambda, g_i) = p_{j_1}(\lambda, g_i)^{\varepsilon_1} p_{j_2}(\lambda, g_i)^{\varepsilon_2}$ and can treat this as a new relation $p_{j_1}(\lambda, g_i)^{\varepsilon_1} p_{j_2}(\lambda, g_i)^{\varepsilon_2} p_j(\lambda, g_i)^{-1} = 1$ holding in **H**. Adding this to the presentation (T1) [see page 11] (and the generators $p_{j_1}(\lambda, g_i)$ and $p_{j_2}(\lambda, g_i)$ if they do not already exist in the presentation (T3)) we can then eliminate $p_j(\lambda, g_i)$ by (T4)'. Continuing in this way, we will eventually obtain a presentation for the subgroup solely in terms of the original subgroup generators.



In the implementation of the algorithm I describe next, a very similar method is employed. It differs in the fact that a single pointer is stored in the coset table instead of a pair. (However, that single pointer does in fact point to this same pair.) The original subgroup generators $h_1, h_2, \ldots, h_s$ are numbered from 2 to s+1 respectively, 1 being reserved for the empty word. When a product $h_{k_1}^{\varepsilon_1} h_{k_2}^{\varepsilon_2}$ is found in the concatenation of h-words we introduce a new subgroup generator $h' = h_{k_1}^{\varepsilon_1} h_{k_2}^{\varepsilon_2}$. This additional subgroup generator is denoted by the next available even number, larger than s+1. This new generator, n say, is defined by

$$n=(k_1+1)^{\varepsilon_1}.(k_2+1)^{\varepsilon_2}$$

and we add relator $(k_1+1)^{\varepsilon_1} (k_2+1)^{\varepsilon_2} n^{-1}$ to the tree by entering $\pm(k_1+1)$ and $\pm(k_2+1)$ (+ if $\varepsilon=1$, - if $\varepsilon=-1$) into the records n and n+1 respectively in the 1-dimensional array TREE. This process works in exactly the same way if one or both the constituents of the product is an extra generator i.e. has number > s+1. Thus, a generator represented by an integer larger than s+1, m say, can always be decoded back to a word in the elements of **Y**, the original subgroup generators. This is done by first looking at the entries in TREE(m) and TREE(m+1). If these are smaller than, or equal to, s+1 we are finished. If not, we look at TREE(TREE(m)) and TREE(TREE(m)+1) to

decode TREE(m), similarly for TREE(m+1). Again if we obtain generators with numbers > s+1 we carry on.

E.g.

m	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	...
TREE(m)						2	3	-2	3	2	6	2	8	-3	6	2	14	...

Here we have original subgroup generators h_1, h_2 (numbered 2 and 3) and wish to decode generator 16 :

$$16 = 2.14 = 2.(-3.6) = 2.-3.(2.3) = h_1 h_2^{-1} h_1 h_2 .$$

I will now give a proof that this process does in fact give a presentation for **H**. This proof uses both the presentations obtained from the Reidemeister-Schreier method and the modified Todd-Coxeter algorithm in Chapter 1, and the information in the last part of that chapter where we attempt to derive one presentation from the other.

Proof

Let $\mathbf{Y}=\{h_1, h_2,\ldots, h_s\}$ and let $\overline{\mathbf{B}}=\{v_{\lambda, g_i}\}$, the set of high-numbered generators appearing in the augmented coset table. Now, let $\mathbf{B}=\overline{\mathbf{B}}\cup\mathbf{Y}$.

If we look at the augmented coset table produced by this version of the modified algorithm, we can see that we can treat the high-numbered generators representing the h-words, $v_{\lambda, g_i}(h)$, as Schreier generators. We shall now denote these high-numbered generators by s_{λ, g_i}.

Thus, from (19) on page 20, we obtain the following presentation for **H** :

$$\langle\, s_{\lambda, g_i}, \ldots \mid s_{\lambda, g_i} = \tau(\bar{\lambda}\, g_i\, \overline{\lambda g_i}^{-1}),\ \tau(\bar{\lambda}\, R\, \bar{\lambda}^{-1}) = 1 \,\rangle. \qquad (*)$$

There are two types of relation obtained from the modified algorithm. One type results from applying each coset representative $\bar{\lambda}$, $1 \le \lambda \le |\mathbf{G}:\mathbf{H}|$, to each relator R, and the second type is obtained from equating each h_k to the word resulting from applying coset 1 to the word representing h_k in the original subgroup generators.

Let $\mathbf{R_1(B)}$ consist of these two sets of relations obtained from the modified algorithm. As shown on page 42, $\mathbf{R_1(B)}$ then includes the second set of relations in (*).

Denote by $\mathbf{R_2(B)}$ the set of relations $\{ s_{\lambda, g_i} = \tau(\bar{\lambda}\, g_i\, \overline{\lambda g_i}^{-1}) \}$ from (*), and set $\mathbf{R_3(B)} = \varnothing$. Then,

$$\mathbf{H} = \langle\, \mathbf{B} \mid \mathbf{R_1(B)}, \mathbf{R_2(B)}, \mathbf{R_3(B)} \,\rangle.$$

If there is a relation in $\mathbf{R_1(B)}$ which allows the largest generator in **B**, say g, to be expressed in terms of smaller generators, we can eliminate g using a Tietze transformation. Then, clearly,

$$\mathbf{H} = \langle\, \mathbf{A} \mid \mathbf{R_1(A)}, \mathbf{R_2(A)}, \mathbf{R_3(A)} \,\rangle$$

where $\mathbf{A} = \mathbf{B} \backslash \{g\}$, and $\mathbf{R_2(A)}$ and $\mathbf{R_3(A)}$ are $\mathbf{R_2(B)}$ and $\mathbf{R_3(B)}$ with g eliminated by the same Tietze transformation that eliminated it from $\mathbf{R_1(B)}$.

If g cannot be eliminated in this way we add the tree relation which expresses g in terms of smaller generators to $\mathbf{R_1(B)}$, and use this to eliminate g. Here there are three cases which we need to consider.

(i) $g = g_1 g_2$ where $g_1, g_2 \in \mathbf{B}$

In this case the relation $g = g_1 g_2$ is theoretically deducible from the other relators in the presentation, i.e. deducible from $\mathbf{R_1(B)}$, $\mathbf{R_2(B)}$ and $\mathbf{R_3(B)}$, so by adding it to $\mathbf{R_1(B)}$, then eliminating g as before, we obtain

$$\mathbf{H} = \langle\, \mathbf{A} \mid \mathbf{R_1(A)}, \mathbf{R_2(A)}, \mathbf{R_3(A)} \,\rangle \quad \text{where } \mathbf{A} = \mathbf{B} \backslash \{g\}.$$

(ii) $g = g_1 g_2$ where $g_1 \in \mathbf{B}$ and $g_2 \notin \mathbf{B}$

Now $g_2 = g_1^{-1} g$ and we perform a Tietze transformation of type (T3), adding g_2 to **B** and $g_2 = g_1^{-1} g$ to $\mathbf{R_1(B)}$. Then we can eliminate g throughout to get

$$\mathbf{H} = \langle\, \mathbf{A} \mid \mathbf{R_1(A)}, \mathbf{R_2(A)}, \mathbf{R_3(A)} \,\rangle \quad \text{where } \mathbf{A} = \{ \mathbf{B} \backslash \{g\} \} \cup \{g_2\}.$$

(The argument for the case where $g_1 \notin \mathbf{B}$ and $g_2 \in \mathbf{B}$ is similar.)

(iii) $g = g_1 g_2$ where $g_1, g_2 \notin \mathbf{B}$

In this case we put two relators, $g_1 = w_1(\mathbf{Y})$ and $g_2 = w_2(\mathbf{Y})$, which express g_1 and g_2 in terms of the original subgroup generators, **Y**, respectively, into $\mathbf{R_3(B)}$. At this stage we do not have to specify exactly what these words are, but the tree contains

enough information for us to construct them. Now we can eliminate g as before to get

$$\mathbf{H} = \langle\, \mathbf{A} \mid \mathbf{R_1(A)}, \mathbf{R_2(A)}, \mathbf{R_3(A)} \,\rangle \quad \text{where } \mathbf{A} = \{\, \mathbf{B} \backslash \{ g \} \,\} \cup \{ g_1, g_2 \}.$$

We continue in this way, removing the highest-numbered generator occurring in the presentation at each step. Thus, after a finite number of steps, we are left with a presentation solely in terms of **Y**, i.e.

$$\mathbf{H} = \langle\, \mathbf{Y} \mid \mathbf{R_1(Y)}, \mathbf{R_2(Y)}, \mathbf{R_3(Y)} \,\rangle.$$

Now, each relation in $\mathbf{R_3}$ started out as $g_i = w_i(\mathbf{Y})$, and must now have become $\overline{w}_i(\mathbf{Y}) = w_i(\mathbf{Y})$, where $\overline{w}_i(\mathbf{Y})$ is the result of applying successive Tietze transformations to g_i. We can now assume that $\overline{w}_i(\mathbf{Y})$ was our original choice of word for $w_i(\mathbf{Y})$, so $\mathbf{R_3(Y)}$ is empty, i.e.

$$\mathbf{H} = \langle\, \mathbf{Y} \mid \mathbf{R_1(Y)}, \mathbf{R_2(Y)} \,\rangle.$$

At any point in this decoding process we can try to simplify the relations $\mathbf{R_1}$ by searching for relations of the form $w_1 w_2^{-1} = 1$, where w_1 is a shorter string than w_2, and subsequently replacing w_2 by w_1 throughout $\mathbf{R_1}$. [Note: we do not change any of the relators in $\mathbf{R_2}$ or $\mathbf{R_3}$.] If any duplicate relations are found in $\mathbf{R_1}$ we can discard one. This simplification process does not alter the presentation given solely by the relations in $\mathbf{R_1}$.

At the end of the decoding process the set of relations $\mathbf{R_2(Y)}$ contains the relations $s_{\lambda, g_i} = \tau(\bar{\lambda}\, g_i\, \overline{\lambda g_i}^{-1})$ written in terms of the original subgroup generators. In fact, since these relations have not been simplified, they consist of strings of generators in **Y** and their inverses, representing entries in the augmented coset table, i.e. the words v_{λ, g_i} in the ordinary modified algorithm. On page 42 I concluded that these relations are consequences of the others in the presentation when v_{λ, g_i} is substituted for s_{λ, g_i}. Thus, the relations $\mathbf{R_2(Y)}$ are consequences of $\mathbf{R_1(Y)}$ and so

$$\mathbf{H} = \langle\, \mathbf{Y} \mid \mathbf{R_1(Y)} \,\rangle$$

as required. $\Diamond$

I will now describe an implementation of a program called SUBGROUP which carries out this version of the modified Todd-Coxeter algorithm. This was written by E.F. Robertson and D.G. Arrell in 1986 and is an improved version of that described in [2]. The basic underlying coset enumeration program is that described in Section 2.2.

SUBGROUP

Initially two files are opened - one called TREE.DAT which will eventually hold the binary tree used to decode the extra generators introduced, the other called PRESN.DAT into which the final subgroup presentation will be written (still including the extra high-numbered generators). The former is a relative access file, so that in the Tietze transformation phase, any pair of pointers in the tree can be readily accessed without having to read in the whole file. An array TREE is also set up (separately from the large array SPACE). This will hold the binary tree until the enumeration completes.

The initial reading in and processing of the subgroup generators and group relators is almost exactly the same as before. However, involutions are now allocated two columns instead of one to simplify the working, and involutory relators are stored since they are needed to construct the final subgroup relations. A variable GENEND is initialized to be NSGP+2† if there is an odd number of subgroup generators, NSGP+3 if an even number. This will be used to mark the end of the section corresponding to the original subgroup generators in the array TREE.

When the enumeration is carried out two tables are used to hold the information in the augmented coset table. The first, the coset table, is arranged as before, the other - the word table - has as many rows as the coset table but one fewer column. It does not need the first two columns which hold the active coset list in the coset table but instead it reserves a column (the first) to hold pointers to the "coincidence words" which will be described later. This array is part of the initial large array SPACE. Thus, NMAX is now calculated to be

((size of SPACE) - RELEND)/(2*NCOL-1),

so cutting almost by half the maximum number of cosets which can be defined. ENUM

† NSPG is the number of subgroup generators.

is then called with an extra parameter, SPACE(NMAX*NCOL+RELEND+1), which locates the beginning of the 2-dimensional array WRDTBL as described in Section 2.2 for the coset table, SPACE. Each time a new row of SPACE is initialized, the corresponding row of WRDTBL is also initialized.

A variable NEXT is initialized to GENEND+1. This denotes the first available even number which can be used to define a new subgroup generator.

Two new variables LOW and HIGH are also introduced. These keep track of the words built up from concatenating the h-words in the word table when relations or subgroup generators are scanned. LOW is used to store the product (or rather the pointer corresponding to it) when scanning in the forward direction. HIGH is used similarly when scanning in the backward direction. For the scanning of relators, both HIGH and LOW are initialized to 1, the empty word, whereas when scanning a subgroup generator, LOW is still 1 but HIGH is initialized to the number of the subgroup generator. This is basically the same method as that described in Section 1.5, except that the h-words are concatenated each time an entry is looked up in the coset table, not just when a deduction or coincidence is found. Thus we only ever have to concatenate two words at a time.

If the forward scan completes with no new information, the value of NEXT is reduced to the value it had before the scan of this relator took place. This saves space by writing over entries in TREE which do not become incorporated into the word table. If this happens with a subgroup generator, the next subgroup generator is given its number instead i.e. the redundant subgroup generator is passed over. This concatenation of words is done in the subroutine CATCON.

CATCON

First of all we check to see if one of the two words to be concatenated is 1, the empty word. If this is the case, the product is just the value of the other word. If neither are 1, before a new subgroup generator is defined to be their product we check that cancellation does not take place either immediately or one step back.

E.g. if we wish to find the product of high-numbered generators n_1 and n_2 we first check that $n_1 \neq -n_2$. Then we look at n_3=TREE($|n_1|$), n_4=TREE($|n_1|+1$), n_5=TREE($|n_2|$) and n_6=TREE($|n_2|+1$).

If n_1 is positive we check that $n_4 \neq -n_2$.

If n_1 is negative we check that $n_3 \neq n_2$.

If n_2 is positive we check that $n_5 \neq -n_1$.

If n_2 is negative we check that $n_6 \neq n_1$.

If one of these situations is encountered we return the appropriate pointer – n_3, $-n_4$, n_6 or $-n_5$ as the product and nothing is entered into the tree. If none of these is found, the next available even number is defined to be the product as described before and NEXT updated. There is no check to see if we are defining a new generator for a word that already has a generator number assigned to it. This would be a lengthy process, and in practice we do not usually waste a lot of storage space on duplicate entries.

The important part of the program is the storing of new deductions and coincidences in the word table.

Deductions

If on the forward scan of a relator $w_1(g_i)g_p w_2(g_i)$ with coset I we reach as far as $Iw_1(g_i)$=J, but find that $J g_p$ is not yet defined, we have $Iw_1(g_i)$=LOW.J .

Then, on the backward scan, if we reach $Iw_2(g_i)^{-1}$=K but find that $K g_p^{-1}$ is not defined we have $Iw_2(g_i)^{-1}$=HIGH.K .

$$\therefore\ K g_p^{-1} = \mathrm{HIGH}^{-1}.\ I\, w_2^{-1}\, g_p^{-1}$$

$$= \mathrm{HIGH}^{-1}.\ I\, w_1 \qquad \text{since} \quad w_1\, g_p w_2 = 1$$

$$= \mathrm{HIGH}^{-1}\,\mathrm{LOW}\ .\ J\ .$$

Thus, $K g_p^{-1}$=J will be placed in the coset table and HIGH^{-1} LOW in the corresponding place in the word table. The inverse equation $J g_p = \mathrm{LOW}^{-1}$ HIGH.K is also entered in both tables.

Coincidences

When a coincidence is found we also obtain a "coincidence word" which we have to consider.

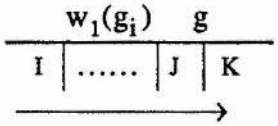
Coincidences can occur in one of three ways. I will examine these in detail

below. Note that in this I will use WRDTBL(I, g_i) to denote the word table entry for I g_i and SPACE(I, g_i) to denote the coset table entry.

(i) On the forward scan through a relation

Here we scan through a relation and reach the end without finding any "holes" in the coset table. A coincidence occurs if we end up at a different coset number from the one with which we started.

E.g. $R_j = w_1(g_i)\, g$ where $g \in X^{\pm 1}$



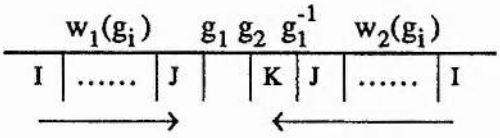
Here $I\, w_1\, g = LOW.K$

$\Rightarrow \quad I = LOW.K$ since $w_1\, g = 1$.

Thus $I \equiv K$ and in this case the final value of LOW, or its inverse, is the coincidence word ICOINWR, depending on which of I and K is the larger. When a coincidence word is found such that ICOINWR . KB = KA with KB > KA, the value of ICOINWR is entered into the first column of the row KB in the array WRDTBL and KB ≡ KA queued in the coset table as in Section 2.2.

(ii) On the backward scan through a relation

E.g.



Here the forward scan has proceeded as far as coset J. However, on the backward scan, $J\, g_1$ has been defined as K. We then define L as $K\, g_2^{-1}$ and enter the deduction $L\, g_1^{-1} = HIGH^{-1}\, LOW.J$ into the table as explained above. However, when we come to enter the inverse equation we discover $J\, g_1$ is already defined, so we have a coincidence.

$\therefore\ I\, w_1 = LOW.J\,, \quad I\, w_2^{-1}\, g_1\, g_2^{-1} = HIGH.L$ and $J\, g_1 = WRDTBL(J, g_1).K$.

Hence $I\, w_1\, g_1 = LOW.J\, g_1 = LOW\ WRDTBL(J, g_1).K$

$\Rightarrow\ I\, w_2^{-1}\, g_1\, g_2^{-1} = LOW\ WRDTBL(J, g_1).K$ since $w_1(g_i)\, g_1\, g_2\, g_1^{-1}\, w_2(g_i) = 1$

$\Rightarrow\ HIGH.L = LOW\ WRDTBL(J, g_1).K$

$\Rightarrow\ L = HIGH^{-1}\ LOW\ WRDTBL(J, g_1).K$.

So, in this case, $ICOINWR = HIGH^{-1}\ LOW\ WRDTBL(J, g_1)$ or its inverse, depending on which of K and L is larger. This is dealt with as in (i) above.

(iii) <u>On equating the rows corresponding to two coincident cosets in the coset table</u>

Here we have $KB \equiv KA$, $KB > KA$. Then,

$WRDTBL(KB, 1).\, KB = KA$ i.e. $ICOINWR = WRDTBL(KB, 1)$,

$KA\, g_i = WRDTBL(KA, g_i).\, JA$

and $KB\, g_i = WRDTBL(KB, g_i).\, JB$.

Now $ICOINWR\ .\ KB\, g_i = KA\, g_i$

so $ICOINWR\ WRDTBL(KB, g_i).\, JB = WRDTBL(KA, g_i).\, JA$

$\Rightarrow (WRDTBL(KA, g_i))^{-1}\ ICOINWR\ WRDTBL(KB, g_i).\, JB = JA$. (*)

Now the ultimate irredundant values of JB and JA, JB' and JA' are calculated as in Section 2.2. If JB is marked as being coincident to JC, say,

$$WRDTBL(JB, 1).\, JB = JC$$

so JB is replaced in (*) by $(WRDTBL(JB, 1))^{-1}.\, JC$. This procedure is repeated until we reach JB'. JA is treated in the same way to find JA'.

Then we have :

$$WRDTBL(JV, 1).....WRDTBL(JA, 1)\ (WRDTBL(KA, g_i))^{-1}\ ICOINWR$$
$$WRDTBL(KB, g_i)\ (WRDTBL(JB, 1))^{-1}.....(WRDTBL(JW, 1))^{-1}.\, JB' = JA'.$$

This new coincidence is added to the queue as before, with the new value of the coincidence word equal to :

$$(WRDTBL(JV, 1).....WRDTBL(JA, 1)\ (WRDTBL(KA, g_i))^{-1}\ ICOINWR$$
$$WRDTBL(KB, g_i)\ (WRDTBL(JB, 1))^{-1}.....(WRDTBL(JW, 1))^{-1})^{\pm 1}.$$

These coincidence words are only incorporated into the main body of the array WRDTBL in one situation i.e. if on processing coincidence KB ≡ KA , KB > KA we find that JA = ∅ and JB ≠ ∅. In this case, SPACE(KA, g_i) is changed to JB and WRDTBL(KA, g_i) to :

WRDTBL(KB, 1) WRDTBL(KB, g_i) if JB ≠ KB and

WRDTBL(KB, 1) WRDTBL(KB, g_i) (WRDTBL(KB, 1))$^{-1}$ if JB = KB.

Note that JA can become zero in one of two situations. Either it has never been defined or it has been defined and subsequently deleted instead of being replaced by another value KX, to avoid having two occurrences of KX in one column. [If we find that JA > JB, JB ≠ ∅ we make SPACE(JB, g_i^{-1}) = ∅ to avoid two occurrences of KA in the same column. Then when we process JA ≡ JB we delete SPACE (KA, g_i) for the same reason and the location SPACE(JB, g_i^{-1}) is filled in with

WRDTBL(JA, 1) WRDTBL(JA, g_i^{-1}). KA
= (WRDTBL(KB, g_i))$^{-1}$ ICOINWR^{-1} WRDTBL(KA, g_i) WRDTBL(JA, g_i^{-1}). KA from (*)
= (WRDTBL(KB, g_i))$^{-1}$ ICOINWR^{-1}. KA since WRDTBL(KA, g_i)$^{-1}$ = WRDTBL(JA, g_i^{-1}).

Now since SPACE(KA, g_i) = ∅ we set SPACE(KA, g_i) = JB and fill WRDTBL(KA, g_i) with ICOINWR WRDTBL(KB, g_i) as originally expected.]

Subgroup Relation Collection

After the enumeration has terminated successfully, the array TREE is first of all written to the file TREE.DAT via the subroutine DUMPTREE, each entry TREE(n) in the array being written to record n. Then the subgroup relations are computed and written to the file PRESN.DAT in the following manner.

Firstly, each of the subgroup generators is scanned with coset 1, looking up each entry in WRDTBL in turn and writing it to an array ARR. At the end of each scan, before the entries in the array are written to PRESN.DAT, the negative value of the subgroup generator is added as the last letter of the relation in ARR. The relation is then written to the file with a 1 at the beginning (the exponent) and 99 at the end, each integer taking up a field of 6 characters. If any WRDTBL entry is 1 it is omitted from the array ARR, and any relation which turns out to be the empty word is in turn omitted from PRESN.DAT.

Note:- We have a problem here with the subgroup generators. If one turned out to be redundant during the defining phase it was not incorporated into the word table and it was "passed over" in the numbering system. However, it is processed in the subgroup relation collection phase. Thus, it is given the number that should belong to the next subgroup generator, so all of the subsequent relations obtained from the subgroup generators will be wrong.

The same procedure is then followed with the group relations, except that the scans are executed with all active cosets and no extra generator is added to the resulting relator when it is written to ARR.

Before the relations are written to PRESN.DAT, the name of the group, ISTAGE (if one was entered), and the value of GENEND are output on the first line. The value of GENEND is vital to the Tietze transformation program as it then knows which generators in the tree are the "extra ones". After all the relations and subgroup generators have been processed with all of the active cosets, the number of relations written to PRESN.DAT is written to the first record of TREE.DAT. This is used by the Tietze transformation program to pinpoint the end of the file. A message including this information is then written to the screen to say that the process is complete, before we return to the main program. Then a message indicating the final size of the tree, given by the current value of NEXT, is output to the screen.

§2.4 A Better Todd Coxeter Program - TC2

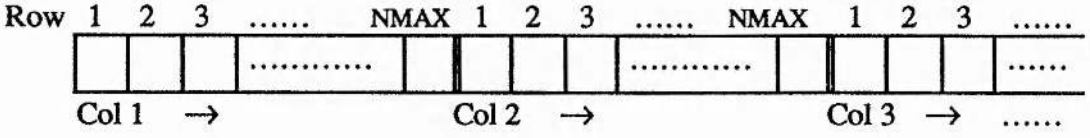
I will now describe the Todd-Coxeter coset enumeration program on to which I grafted the modified algorithm. It is version 2.2A (Feb 1981) of a program designed by G. Havas of ANU, Canberra and programmed in FORTRAN 66 by W.A. Alford - also of Canberra. The version at which I am looking has had a Reidemeister-Schreier process added to it by E.F. Robertson (St Andrews) in June 1984. I will describe this in Section 2.5. The program is running in St. Andrews on a VAX 11/785 and also a SUN 3/260, with a few alterations.

This coset enumeration program implements the Felsch, HLT and Lookahead algorithms and the user has a great deal of control over the way in which these are

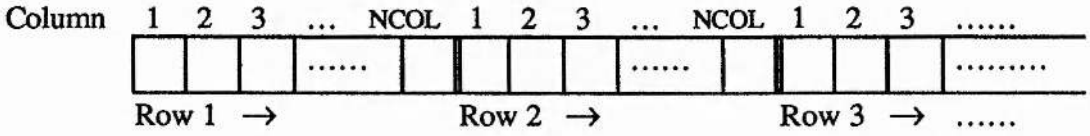
carried out. There are many other extra facilities which exist. For example it is possible to add in more subgroup generators and relators when an enumeration has completed or stopped, and continue the enumeration from the existing coset table. There are also commands to delete specific subgroup generators and relators within the program - although, obviously, the enumeration does have to be started again from scratch in this case. You can also find the normal closure of the subgroup generators, find the coset representatives, or print the permutation representation of the group generators from the coset table. Another very useful feature is that partial or complete coset tables and the values of all associated variables can be saved to a file. These can be read back in later and the enumeration resumed. Thus, this program is much more of a "coset enumeration package" than TC1.

In many respects the variable names and the data structures are very similar to those used in TC1. The code in the HLT routine is almost identical line for line to that in the subroutine ENUM in TC1. Thus, TC1 was almost certainly the enumeration program used as a basis for TC2.

Like TC1, all of the subgroup generators, the group relators, the column translation tables and the coset table are stored in one large array - this time called Y. In the subroutines, part of Y is in common with the 2-d array SPACE, which is again the coset table. SPACE is adjustably dimensioned (NCOL, NMAX) with its first location at Y(FRONTY). It is worth noting that the columns are indexed first in SPACE, then the rows. This has been done because FORTRAN stores arrays a column at a time, one after the other.



Thus, if we stored the coset table in the conventional manner we would use up a lot of unnecessary space because we would have to leave space for NMAX rows in each column, whether or not we were going to define that many cosets. Storing the transpose of the coset table means that as we initialize a row at a time (now a column in the array) all of the defined entries will be at the beginning of the 1-d array.



We can now use the large number of contiguous locations at the end of SPACE as temporary storage and can also increase NMAX within the program without changing the position of any entries in the array. However, I will continue to discuss the coset table in its usual format, and not in the manner in which it is stored.

The enumeration is usually done from the subroutine OPT which controls all the functions of the program, except reading in the initial data. Some variables are initialized with default values in BLOCK DATA, although most of these can be overridden by the user. Data can be input interactively, although it is usually much easier to do this from a data file. This can be created using a program such as IN running on the machines at St Andrews. The subgroup generators and the relations are read in, using the subroutine READIN. Unlike TC1, these are read in as a list of generators (numbered from 1 upwards) and their inverses (-1, -2 etc.) prefixed by an exponent, which must be ≥ 1. It is possible to have more than 9 group generators if the input format is changed. A non-zero digit in the last input character signifies continuation on to the next line. The length of the word is then calculated and the subgroup generator or relator stored at the front of Y in the format

length of word exponent word

When all the subgroup generators and relators have been read in, we calculate the number of columns needed for the coset table. Like TC1 only one column is allocated to an involutory generator. We do not add any extra columns to hold the list of active cosets and the coincidence queue, instead, they are kept in the first two columns of the normal coset table. This implements an idea of M.J. Beetham [3] and will be explained later. The method requires the first two columns of the coset table to belong either to a non-involutory generator and its inverse, or to two involutory generators. Thus, the column numbers allocated to the generators are not always in the order one would expect .

The first non-involutory generator to be found is given column 1, and its inverse, column 2. If every generator is an involution, the first two are allocated columns 1 and 2, or if there is only one, it is assigned two columns.

The group generators in the subgroup generators and relators are then translated into their associated column numbers. Any involutory generator relators are marked with a negative flag in the exponent. This is for the HLT implementation and stops the relator being used for scans - any deductions to be gained from it already

exist in the coset table since we have used only one column for an involution and any definition which needs to be made in it will be done when checking that row of the coset table for "holes".

We then assemble the pointers to essentially different positions using the routine APEDP if the variable FELSCH is TRUE. Thus, if a relator is stored as 3 3 1 2 1 rather than 9 1 1 2 1 1 2 1 1 2 1 we find fewer different positions in the first configuration for each of the generators than we do in the second. Then, when scanning relators in Felsch with the deduction list, the process will terminate much more quickly in the first case. This is not done for involutory relations. The subgroup generators and relations and the table of pointers for the Felsch method (if they exist) are then shifted to the end of the array Y. FRONTY then marks the beginning, and PTEND the end, of the available space in Y for the coset table.

For the Felsch method, the deduction list starts at Y(PTEND) and extends towards the front of the array. If it reaches the coset table, any new deductions are discarded. However, two locations are always kept above the coset table to hold one deduction. These are <u>never</u> overwritten by the coset table if FELSCH=TRUE. Thus, the maximum possible number of rows available for the coset table is calculated to be MAXROW, where

$$\text{MAXROW} = \begin{cases} (\text{PTEND} - \text{FRONTY} - 1) / \text{NCOL} & \text{for Felsch} \\ (\text{PTEND} - \text{FRONTY} + 1) / \text{NCOL} & \text{for HLT.} \end{cases}$$

As I said before there are many variables which can be used to control the enumeration. These are either read in with the initial data or changed via certain commands in subroutine OPT. I will describe the more important ones here.

MAXTAB Like TC1, the user can input a value to limit the number of cosets which can be defined to the value of MAXTAB, if MAXTAB > $\emptyset$. If MAXTAB < $\emptyset$, -MAXTAB rows are reserved at the end of the coset table for use with some of the facilities available in SUBROUTINE OPT. The maximum possible number of rows in the coset table, MAXROW, is calculated remembering to leave 2 locations for the deduction queue if FELSCH=TRUE. Thus, the new maximum number of rows allowed, MAXROW = min(MAXROW, MAXTAB) if MAXTAB > $\emptyset$ or MAXROW = min(MAXROW, MAXROW+MAXTAB) if MAXTAB < $\emptyset$.

NLOOK If this variable is positive it tells us the maximum number of lookaheads to carry out in the HLT enumeration, provided that at least 10% of the array SPACE is recovered in each lookahead. If NLOOK=$\emptyset$, $-50 \leq$ NLOOK ≤ -98 or NLOOK < -100 by default we get a maximum of three lookaheads with at least 10% of the array space needing to be recovered in each lookahead. If $-49 \leq$ NLOOK ≤ -1, |NLOOK| is the percentage of the array SPACE that must be recovered in a lookahead for the enumeration to continue. If NLOOK=-99 no limits are placed on the number of lookaheads or the amount of space to be recovered in a lookahead, and if NLOOK=-100 no lookaheads are done. Thus, by making NLOOK=-100, we can do a straight HLT.

NLOOK is used in the Felsch algorithm to specify how many group relators are to be included as subgroup generators. This option is here, because sometimes fewer cosets need to be defined if the group relators are also subgroup generators. If NLOOK is greater than the number of relators, all relators are included as subgroup generators; if NLOOK $\leq \emptyset$, no relators are included.

CNTRL This variable controls the type of enumeration done and the method of collecting and reusing space freed by coincidences. It also allows access to the subroutine OPT or enables you to finish a job from the main program. The possible values are as follows :

±1 Felsch with a linked list to mark freed space.
±2 Felsch with compaction to collect together freed space.
±3 HLT plus Lookahead with a linked list.
±4 HLT plus Lookahead with compaction.
±5 Enter Option Routine.
±9 End of job.

To use the interactive facilities of the program you have to carry out the enumeration from the subroutine OPT. Commands in OPT are given as a combination of 1 or 2 letters followed by a number - the parameter, PARMTR. A GO TO statement then sends you to the correct part of OPT where the required task is carried out , often calling other subroutines in the process. When the task is finished, it goes back to the

beginning of the routine and asks for the next option to be input. The command EX stops the execution of the program.

The HLT method is carried out in the subroutine ENUMTL, Felsch in the subroutine ENUMFL. Each time a new coset is initialized, NALIVE is checked to see if it has increased or decreased by 1000 since the last time the enumeration statistics were printed. If it has, the value of NALIVE is output together with the maximum number of cosets active at any one time, MAXCOS, and the total number of cosets defined so far, TOTCOS.

When we have CNTRL=±2, ±4 and need to define another coset, but cannot define any more in sequence, we compact the table using the subroutine COMPAC. First we check the value of LOWCOM. At the beginning of the enumeration it has the value of BACKY, the last location in Y. If any coincidences have occurred it is given the value of the smallest coset number which has been eliminated. If LOWCOM > MAXROW there have been obviously no coincidences and we must go into the lookahead phase, if HLT, or stop with a message saying that there is not enough space left, if Felsch. If LOWCOM < MAXROW we check to make sure that the coset LOWCOM is still inactive. If it is active we check the next coset in numerical order and so on until we find a coset which has been eliminated, ROW. We then look for the first active coset, I, larger than this. All entries in the deduction queue referencing row I are changed to its new row number, ROW. Then we look at J=IABS(SPACE(COL, I)) for COL=1,..., NCOL. If J>∅ and J≠I we find the column number of the inverse of the generator with number COL. We shall call this K. Then SPACE(K, J)=ROW is filled in to the coset table. If J=I, the value of J is changed to ROW. In both cases SPACE(COL, ROW)=ISIGN(J, SPACE(COL, I)) is filled in to the table. This entry just gives SPACE(COL, ROW) the absolute value of J with the same sign as SPACE(COL, I). This ensures that any negative flags are passed on. For each redundant coset we encounter smaller than KN, we increment a counter KNS by 1, and before we return from the subroutine we can compute the new value of KN to be KN-KNS. Similarly, if we are carrying out a Felsch enumeration, we have to keep track of the number of rows to subtract from ROWMIN, the row in which the gap of length one or minimum length is stored. Other variables such as LASTPS and NEXTDF are also updated.

Note that if COMPAC is called in the middle of tracing out a relation, the tracing of that relation is always restarted from scratch with KN in case the values of IFRONT and IBACK have changed.

Some of the subroutines called by ENUMTL and ENUMFL are identical, so I will describe them in detail for the HLT method.

§2.4.1 HLT

If we are carrying out the HLT method we call subroutine ENUMTL with the actual parameters (RENTER, NMAX, NCOL, SPACE) from inside subroutine OPT, (RENTER, NMAX, NCOL, Y(YFRONT)) from the main program. RENTER is non-zero if we are resuming an enumeration from a partial or complete coset table, zero if we have to start from scratch.

ENUMTL

Two sets of negative flags are used in the coset table. A negative flag in column 1 indicates that this coset has become redundant and the row is available for reuse. A negative flag in column 2 indicates that this coset has been completely processed in the defining phase or completely processed in the lookahead phase (i.e. it is "closed"). FREEL denotes the top of the list of cosets available for reuse and SPACE(1, I) is used as the link in this list i.e. -SPACE(1, FREEL) is the second coset available for reuse if the value of FREEL is non-zero. FREELM is used to keep track of the minimum coset number which has been reused.

A variable INDEX1 is used to detect whether each column in the coset table contains at least one entry if NALIVE falls to 1 in the coincidence routine. If this is the case, INDEX1 is set to TRUE, INDEX is given the value 1 and the enumeration terminated. This cuts out a lot of unnecessary processing in the coincidence routines.

Instead of the integers -1, 0, +1 being used to denote the subgroup generation phase, the lookahead phase and the defining phase respectively, two logical variables SUBGRP and LOOKAH are used in TC2.

The subgroup generators are processed in the subroutine APPLY, which is called with one subgroup generator at a time.

APPLY

This is very similar to part of ENUM in TC1. It applies HLT with coset HCOSET to a word starting at Y(BEG) and finishing at Y(END) with exponent at Y(BEG-1), defining cosets to complete the tracing out of the word if needed. This

process is slightly more complicated than that in TC1 if the exponent is > 1. In this case we have to ensure that we scan through the word the required number of times. The total number of characters still to be scanned in the expanded word is monitored with the variable COUNTR. Like ENUM new cosets are only defined on the backward scan. When a new coset is defined, it takes the value of NEXTDF if NEXTDF is smaller than MAXROW and NEXTDF is incremented by 1. If NEXTDF > MAXROW, we can define no new coset numbers to extend the coset table. If FREELM=1 we have no free space at all in the table, so we set OVERFL=TRUE and RETURN. If there is space in the coset table, our next step depends on the method of reusing freed space. If CNTRL=±4 we call subroutine COMPAC to compact the table, otherwise we let the new coset be FREEL and update FREEL to -SPACE(1, FREEL). FREELM is then changed to the minimum value of FREEL and FREELM.

If compaction is carried out in the middle of processing a subgroup generator, we cannot carry on with the processing afterwards as the values of IFRONT and IBACK may have changed. Therefore, we RETURN and call APPLY again. If a forward or backward scan completes with a coincidence the routine COINC is called with IFRONT and IBACK, the coset numbers reached in the forward scan and backward scan respectively.

ENUMTL (continued)

After each subgroup generator has been processed, we check to see if either INDEX1 or OVERFL=TRUE. If we have the former the enumeration is terminated, and if we find the latter we enter the lookahead phase.

When the the subgroup generators have all been processed we start on the group relators. The relation defining phase is exactly the same as the enumeration code in APPLY, only it is carried out with KN equal to each defined coset number in turn. However, if we have an involutory relator (i.e. exponent < ∅) it is skipped over - if it does not close, the required coset will be defined in the coset table anyway. When KN has been applied to all group relators, we look at row KN in the coset table to make sure that all entries are defined. If there are any gaps, we define cosets to fill them, and increment KN by 1. When KN becomes greater than FREELM or a coset number less than KN disappears through coincidences, a variable KNL is set to TRUE. This tells us that when KN reaches NEXTDF in the defining phase, we must set KN to

FREELM (if it is non-zero) and check that all cosets defined in freed space before the current value of KN are also closed. At this point we set FREELM to $\emptyset$ and KNL to FALSE. When we reach KN=NEXTDF again, if FREELM is still zero, the enumeration has completed. Otherwise, we must make KN equal to the new value of FREELM and repeat the process. When we go into the lookahead phase all rows less than, or equal to, MAXROW have been defined. We take LCLOSE=KN-1. This is the last coset to have closed in the defining phase, and, after the lookahead, KN will start again at LCLOSE+1.

A variable CLOSED is used to check whether a row is closed in the lookahead phase. It has the value TRUE until a gap is found in a relation, in which case it becomes FALSE. If CLOSED=TRUE and there are no gaps in the coset table for coset KN, SPACE(2, KN) is negated to show that coset KN is closed.

In the lookahead KN is incremented by 1 until it reaches NEXTDF, at which point the lookahead is deemed to be complete and we return to the defining phase.

Note that the enumeration never completes on the lookahead phase, only after thorough checking in the defining phase. When the coset table is complete, the index and the values of NALIVE, MAXCOS and TOTCOS are written out. Then the negative flags are removed from column 2 and the routine exited.

§2.4.2 Felsch

To carry out the Felsch method we call the subroutine ENUMFL with the same parameters as ENUMTL. There is a variable called STOPLP, especially for Felsch, which controls the different strategies used to decide where the next coset should be defined. This can either be read in with the initial data or changed in subroutine OPT. If no value is supplied by the user, a default value of 5 is used.

If the value of STOPLP is positive, and during a relator scan a gap of length 1 occurs (i.e. only one more coset need be defined to close this relator cycle), this gap is noted. Provided no subsequent deductions fill this gap, (in which case we note the next gap of length 1) or a coincidence occurs (when the noting of gaps starts from scratch), the next coset to be defined will be defined to fill this gap. This process can cause infinite looping on awkward examples, so we insist that STOPLP*KN is at least NALIVE before a coset is defined to fill a gap of length 1. If it is not, we define our

next coset in the first available space in or after row KN. This ensures that the coset table is being steadily filled up.

If STOPLP is negative, the minimum gap found in the relator scan is used to determine where the next coset will be defined, again insisting that |STOPLP|*KN ≥ NALIVE. Thus, if STOPLP=±1, the original Felsch Algorithm will be used, since we will always make our definitions in the coset table.

Each time a definition or a deduction is written into the coset table, this information is also inserted into the deduction queue, using the subroutine NTDEDT. The column and the row of SPACE in which the deduction has been written are noted in the locations Y(NDED) and Y(NDED-1) respectively, and NDED is then reduced by 2. The first deduction in this queue is kept at Y(PTEND) and Y(PTEND-1). The location in Y of the current deduction which we are processing is given by the value of NDEDD and, after each deduction has been processed, NDEDD is reduced by 2. If there is not enough space for any more deductions on the queue, i.e. NDED-2 < LASTPS, we shift the queue back up to PTEND writing over any deductions which have been processed or disappeared due to coincidences. When no further space is available for the deduction queue, any new deductions found are discarded. This should not affect the result of the enumeration, only the time taken for it to complete. It is possible theoretically that these discarded deductions could help the enumeration to complete, whereas without them it doesn't; however, if the coset table is nearly full, it is very likely that the enumeration would not complete successfully anyway.

ENUMFL

Depending on the value of STOPLP, we make MINGPL=TRUE if we are to use the minimum gap strategy, FALSE if we are to use gaps of length 1.

The subgroup generators are processed using APPLY in exactly the same way as those in the HLT method, except that now all definitions and deductions are added to the deduction queue using NTDEDT. Remember that depending on the value of NLOOK we may also be processing some (or all) of the group relators as subgroup generators. Thus, when we have finished the subgroup generation phase we could have a sizeable deduction queue. The group relators are then processed from the deduction queue. The row and column of each deduction is extracted from Y(NDEDD-1) and Y(NDEDD) respectively. If Y(NDEDD-1)=∅, this deduction has been

deleted after the coincidence processing. If this is the case, the next deduction is obtained from the queue. This deduction is then tried in all possible places in each relator, these being found from the table of pointers to essentially different positions.

First of all, we scan backwards through the relator from the deduction until we find a gap in the coset table. If we reach the beginning of the word, we continue the backward scan from the end of the relator with the same coset.

If the backward scan completes, we get a coincidence, and if this is non-trivial we call COINC.

If the backward scan does not complete, we begin the forward scan, starting at our original deduction. If it completes, we have a coincidence, and COINC is called, (assuming it is not trivial).

If the forward scan does not complete, we check to see if we have a deduction, in which case this, and its inverse, is filled into the coset table and the information added to the deduction queue.

If there is no deduction, we check to see if we have found a gap of length 1 or a minimum gap, depending on the value of MINGPL. MINGAP holds the length of the current minimum gap i.e. the number of holes in that row of the relator table which have to be filled. MINLEN is a logical variable which indicates whether or not a gap of length 1 or a minimum gap has been stored. If it is TRUE, the location in SPACE referring to one end of the gap is referenced by the values of the variables ROWMIN and COLMIN. If MINLEN=TRUE, we always check to make sure that this gap has not been filled by a deduction. If it has, we store our new location in ROWMIN and COLMIN.

When we have tried this deduction in all possible places in each relator, we obtain the next deduction from the deduction queue. When there are no more deductions to be dealt with, the coset table is searched to find the first empty space in a row, greater than or equal to KN. Each time we increment KN, we check SPACE(1, KN) is greater than $\emptyset$ i.e. KN has not been eliminated during coincidence processing. This empty location is the first to be filled if we have no minimum gap or gap of length 1 noted. Each time a definition or deduction is made in the relator defining phase, we start processing the deduction queue again.

This process of making a definition, then processing it and any consequential

deductions, is continued until KN reaches NEXTDF. Then we start the process again with KN=FREELM (making FREELM=$\emptyset$) and make sure that all cosets which have been redefined have no gaps in their rows. When we reach NEXTDF again, assuming no more coincidences have occurred and FREELM still equals $\emptyset$, the coset table is complete, otherwise we start again at the new value of FREELM.

§2.4.3 Coincidence Processing

The coincidences found when running APPLY, ENUMFL and ENUMTL are processed using three main subroutines - COINC, PCL12 and NTCOIC. PCL12 processes the first two columns of the initial coincidence and any other coincidences resulting from these, putting all of the coincidences on the coincidence queue. COINC processes columns 3 to NCOL of all coincidences on this queue, calling PCL12 for each new coincidence it encounters. NTCOIC notes any new coincidences arising from PCL12 and stores them until they can be processed by PCL12.

In the coincidence routines the variable names used for the two coincident cosets currently being processed are HIGH and LOW, obviously HIGH being the larger of the two coset numbers. SPACE(2, HIGH) is used to link the coincidence queue together and SPACE(1, HIGH) holds the coincident coset -LOW. This minus sign is a redundancy flag, and, when row HIGH has been fully processed, SPACE(1, HIGH) is then used to link row HIGH to the top of the free space list. As in TC1, QHEAD points to the top of the coincidence queue, QTAIL to the bottom.

Since the coincidence queue is kept in the first two columns of the coset table, this is the reason that we have to process the first two columns of any coincidence before we queue it.

PCL12

This is the most complicated part of the coincidence routine. We set aside four locations - LOW1S, HIGH1S, LOW2S AND HIGH2S to store the coincidences of which we still have to process the first two columns. We never have any more than two coincidences in this category, for reasons which I will explain later.

We look at our initial coincidence HIGH≡LOW and compute the ultimate irredundant values of the two cosets. First of all we check the entry in the first column of row HIGH. If it is $< \emptyset$, we look at the first entry in the row belonging to this new

coset. We continue in this way until we find the smallest coset number that HIGH is coincident to, HIGH′, and fill -HIGH′ into the first column of every coset in this chain. Then we find LOW′ in the same way and fill -LOW′ into the first column of the coset table for each member of the LOW chain.

If LOW′=HIGH′ nothing more is done and we obtain the next coincidence to process from the locations LOW1S, LOW2S, HIGH1S, HIGH2S (if they are non-empty).

If LOW′≠HIGH′ we make a note of the first two entries in row HIGH′ by making HIGH1=SPACE(1, HIGH′) and HIGH2=IABS(SPACE(1, HIGH′)) and then mark HIGH′ coincident to LOW′. We then look at the consequences of column 1 of rows LOW′ and HIGH′.

If HIGH1=∅, we do nothing.

If HIGH1≠∅, J=Y(INVCOL+1) i.e. J is the column containing the inverse of the generator in column 1. This, by the construction of the column numbering system, must be 1 or 2. If HIGH=HIGH′ we change this value to LOW′, otherwise we delete SPACE(J, HIGH1) instead of replacing it by LOW′, to avoid the possibility of having two occurrences of LOW′ in the same column.

Now LOW1=SPACE(1, LOW′).

If LOW1=∅, we insert HIGH1 into SPACE(1, LOW′) and note this deduction. If LOW1≠∅, we have a coincidence LOW1≡HIGH1. We replace LOW1 by LOW′ if LOW1=HIGH′ (and replace SPACE(1, LOW) by LOW′), then we call NTCOIC. This first checks that we have not already found this coincidence i.e. LOW1≠LOW1S, HIGH1≠HIGH1S and LOW1≠LOW2S, HIGH1≠HIGH2S. If it has already been noted, we do nothing. If it has not been noted, we queue it in LOW1S and HIGH1S if LOW1S=∅, or in LOW2S and HIGH2S otherwise.

Then, if SPACE(1, LOW)≠∅, we check SPACE(J, SPACE(1, LOW)). If it is zero we check SPACE(1, LOW). If this equals HIGH′, we go on to process column 2, if not, we make SPACE(J, SPACE(1, LOW))=LOW′ trying to preserve any negative flagging that exists on column 2, if possible.

Column 2 is treated in the same way as column 1, with the subscripts 1 and 2 reversed where necessary. When this has been done, we reduce NALIVE by 1, and if NALIVE=1, we check that each column in the coset table has at least one entry defined, in which case INDEX1=TRUE.

If the index is not 1, we remove coset HIGH′ from any cosets stored up to be processed i.e. we check that none of LOW1S, HIGH1S, LOW2S, HIGH2S is equal to HIGH′. If one is, it is replaced by LOW1′.

We then take the next pair of coincident cosets and process their first two columns. These are LOW1S and HIGH1S, if LOW1S is non-zero, otherwise LOW2S and HIGH2S. Once these have become the new values of LOW and HIGH, the appropriate variable, LOW1S or LOW2S, is set to zero. If LOW1S and LOW2S are both found to be zero, we have processed the first two columns of all pending coincidences, and can return to COINC to process columns 3 to NCOL of each coincidence on the queue.

This method ensures that when we process the first two columns of any coincidence, other than the first, we find at most one other coincidence which takes its place in the storage variables LOW1S and HIGH1S or LOW2S and HIGH2S. This process works slightly differently in the case where we have two involutions assigned to the first two columns of the coset table, rather than one generator and its inverse.

(i) Non-involutory generators

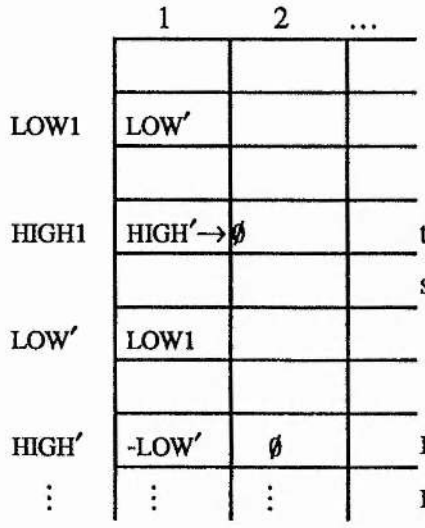
After processing HIGH′≡LOW′, we then look at HIGH1≡LOW1, if neither are zero. Now, processing the first column of this gives a coincidence HIGH11≡LOW11, but the second column gives HIGH′≡LOW′, or rather LOW′≡∅, since HIGH′ would have been replaced by ∅ to avoid having two occurrences of LOW′ in the same column.

	1	2	...	
LOW1	LOW11	LOW′		
HIGH1	HIGH11	HIGH′→∅		to avoid two occurrences of LOW′ in the same column.
LOW′	LOW1	LOW2		
HIGH′	-LOW′	∅		HIGH1=original entry in SPACE(1, HIGH′)
⋮	⋮	⋮		HIGH2=original entry in SPACE(2, HIGH′)

Thus, we only get one coincidence.

(ii) Involutory generators

Here, processing the first column of HIGH1≡LOW1, we get :



to avoid two occurrences of LOW′ in the same column.

HIGH1=original entry in SPACE(1, HIGH′)

HIGH2=original entry in SPACE(2, HIGH′)

Since SPACE(1, HIGH1) would have originally equalled HIGH′, it will have been changed to zero to avoid two occurrences of LOW′ in the same column. Thus we obtain no deduction from the first column of HIGH1≡LOW1, so we again have, at most, one coincidence to queue. Processing HIGH2≡LOW2 works in the same way, yielding, at most, one coincidence. It can be seen that any other resulting coincidence follows suit. This process is explained in essence in [3], but there a different data structure is suggested for storing the coincidence queue.

COINC

After calling PCL12 to process the initial coincidence HIGH≡LOW, columns 3 to NCOL are similarly processed. Each time we find a potentially new coincidence HIGHI≡LOWI, where HIGHI and LOWI are the values in column I of rows HIGH and LOW respectively, we call PCL12 and process columns 1 and 2 of this new coincidence and all ensuing ones. When we return from PCL12, we continue with the initial coincidence HIGH≡LOW.

When we have compared the last of these cosets' columns, we extract the next values of HIGH and LOW from the head of the coincidence queue. Each time a coincidence has been processed, LOWCOM is changed to the minimum of HIGH and LOWCOM, so that when the coincidence queue is exhausted, LOWCOM is the smallest

coset which has been eliminated due to coincidences. When the queue is exhausted, we check coset KN to make sure it is still active. If it is not, we reduce it by 1 until we find an active coset.

Before we return to the enumeration routines, if FELSCH=TRUE we delete all entries in the deductions queue which reference eliminated cosets. This is done by looking up the first column in the coset table of each row appearing in the deduction queue, and if it is negative, this row value is replaced by $\emptyset$ in the deduction queue.

§2.2.4 Compaction versus Linked Lists

The two different methods in which the space freed by coincidences is collected and reused can have a profound effect on the statistics for the enumeration. This is basically because coset numbers are defined, and subsequently processed, in a different order.

For HLT with compaction all cosets are checked for closure in the order in which they are defined. Usually, cosets which are defined near the beginning of an enumeration will have a better chance of closure than those newly defined, and fewer new cosets are needed to fill gaps in relator scans or in their row in the coset table. However, with HLT using linked lists, if a coset I has been reused, where I is larger than KN, this value of I will be used to try and close rows in the relator and coset tables before other cosets > I which were defined previous to it. Intuitively, this means that we may have to define more redundant cosets. However, this is not always the case, as we may find some important deduction or coincidence much sooner, allowing us to collapse the coset table more quickly.

In Felsch this difference in the processing of freed space does not affect the results significantly, if the minimum gap or gap of length 1 strategies are used. However, if we resort to filling holes in the coset table most of the time, (either because IABS(STOPLP)*KN < NALIVE, STOPLP=±1 or because the minimum gap or gap of length 1 has already been filled), the order in which the cosets have been defined will matter.

We increment KN by 1 each time a row in the coset table has been closed. However, again, it is not until we reach MAXROW that we reset KN to FREELM and look at the rows which were reused after they were passed by KN. Thus, in some

peculiar examples, just because KN*IABS(STOPLP)>NALIVE, it does not mean to say that the first KN rows of the coset table are anywhere near full! Perhaps some count should be kept of all cosets less than KN which are made redundant, and this taken into consideration when we decide whether or not to define in the coset table, instead of in the minimum gap or gap of length 1.

If the enumeration runs out of space often and has to compact, a lot of time is wasted, especially if the coset table is large and there have been coincidences near the beginning. Thus, the methods using linked lists are usually quicker.

§2.4.5 Alterations to TC2

(1) In 1989 C. Sims discovered a bug in the coincidence routines when using the Felsch method [22]. This bug doesn't just prevent some coset tables from closing when they should, but for certain examples can result in closed coset tables with the wrong index. Basically, he discovered that in certain circumstances, entries are changed in the coset table, but this new information is not placed on the deduction queue. It is possible, then, for the enumeration to terminate with a complete coset table; however, if this deduction which was not put on the deduction queue was now to be tried in all essentially different places in the relators, coincidences are found which collapse the coset table.

This error happens if SPACE(I, LOW)=HIGH for any column I when processing the coincidence HIGH≡LOW. In this case, SPACE(I, LOW) is changed to LOW but the new information is not placed on the deduction queue. [Of course it may already exist there, if (LOW, I) had been put there previously and has not yet been processed.] Usually this does not cause a problem, because we will eventually call NTDEDT with this deduction when we process the coincidence HIGHI≡LOWI or HIGHJ≡LOWJ (where $J=I^{-1}$) or coincidences resulting from these.

However, in the example C. Sims has constructed, this information is never placed on the deduction queue. In his example, as well as SPACE(I, LOW)=HIGH, SPACE(I, HIGH)=LOW.

We will look at this in more detail where I is a non-involutory generator. Let the inverse of I be J. Since SPACE(I, LOW)=HIGH, SPACE(J, HIGH)=LOW. Similarly, SPACE(J, LOW)=HIGH. Thus we have the following situation :

	...	I	J	...
⋮		⋮	⋮	
LOW	...	HIGH	HIGH	...
⋮		⋮	⋮	
HIGH	...	LOW	LOW	...
⋮		⋮	⋮	

(1)

Processing column I of LOW≡HIGH, we set SPACE(J, HIGHI)=SPACE(J, LOW) to zero. Then, since LOWI=HIGH, we put SPACE(I, LOW)=LOW and LOWI=LOW. We then call PCL12 with the coincidence LOWI=HIGHI. However, since both LOWI and HIGHI have the value LOW, nothing is done. Then we look at SPACE(J, LOWI) =SPACE(J, LOW) and since it is zero, we write in the value LOW. At this point we have the following table :

	...	I	J	...
⋮		⋮	⋮	
LOW	...	LOW	LOW	...
⋮		⋮	⋮	
HIGH	...	LOW	LOW	...
⋮		⋮	⋮	

(2)

Then we process column J. We temporarily set SPACE(I, LOW) to zero. Again, when we call PCL12 with LOWJ and HIGHJ, nothing is done because they both have the value LOW. We then write LOW back into SPACE(I, LOW). Thus, we again end up with the coset table (2) above. These deductions in SPACE(I, LOW) and SPACE(J, LOW) are not placed on the deduction queue during the processing of any columns in this coincidence, so, unless LOW becomes coincident to yet another coset, this information will not be incorporated into the deduction queue.

If I is an involution, the inverse of I, namely J, is I, and we have the following coset table:

	...	I	...
⋮		⋮	
LOW	...	HIGH	...
⋮		⋮	
HIGH	...	LOW	...
⋮		⋮	

(3)

Processing column I of LOW≡HIGH, we again set SPACE(J, HIGHI)= SPACE(I, LOW) to zero. Then LOWI=∅, so we put SPACE(I, LOW)=HIGHI=LOW and call NTDEDT.

Thus, the problem only seems to appear for non-involutory generators when LOWI=HIGH and HIGHI=LOW. I have looked at the code very carefully and this appears to be the only situation where information is omitted from the deduction queue. I decided that the easiest thing to do was to insert the statement CALL NTDEDT(LOW, I) each time SPACE(I, LOW) is changed from HIGH to LOW. This occurs twice in PCL12 and once in COINC.

When G. Havas came up with a "bug fix", he suggested that the three statements SPACE(I, LOW)=LOW simply be omitted. This works in a roundabout way. Going back to coset table (1), and processing column I, everything takes the same value as before except that SPACE(I, LOW)=HIGH. After a futile call to PCL12 with LOWI=LOW and HIGHI=LOW, LOWI is again given the value HIGH (since SPACE(I, LOW)=HIGH) and since SPACE(J, LOWI)=SPACE(J, HIGH)≠∅, nothing more is done. At this point our coset table now looks like :

	...	I	J	...
⋮		⋮	⋮	
LOW	...	HIGH	∅	...
⋮		⋮	⋮	
HIGH	...	LOW	LOW	...
⋮		⋮	⋮	

(4)

Then we process column J. We set SPACE(I, LOW) to zero. Since LOWJ=∅ we put SPACE(J, LOW)=HIGHJ=LOW and call NTDEDT. Then, LOWJ=LOW and since SPACE(I, LOWI)=SPACE(I, LOW)=∅ we put SPACE(I, LOW)=LOW.

Thus, the deduction has now been noted. It is interesting to note that in the subroutine COINC of TC1, this offending line SPACE(I, LOW)=LOW does not appear ! It must have been introduced into later versions of the program by someone thinking that it improved the efficiency of the algorithm.

(2) While examining the code in ENUMTL I found a bug which, although it doesn't affect the final result, can affect the efficiency of the enumeration. This bug occurs in the running of the HLT method with Lookahead when linked lists are being used to collect freed space. As the code stands, coset numbers < KN are not used in the relator scans in the lookahead phase, so if FREELM<KN (i.e. KNL=TRUE), the coset numbers between FREELM and KN, which have been reused but not subsequently closed, are not applied to the relators. This can cause problems when KN is large and a substantial number of coset numbers less than KN have been reused, since only a fraction of the potential deductions and coincidences are obtained from the lookahead. This may prevent an enumeration from completing when there is in fact enough space, if all available information was picked up and utilized.

Thus, for each new lookahead I reset KN to the value of FREELM. I also changed the program so that if coset FREELM was found to be closed, FREELM was incremented by 1. This could save time if there are many lookaheads, since the known closed cosets do not have to be retested for closure each time.

(3) Another bug which exists in ENUMTL affects the enumeration when the lookahead phase is entered part way through the subgroup generation phase. This can happen realistically if there are a large number of subgroup generators. The lookahead is applied to the remainder of the subgroup generators, then to all the group relators with each active coset in turn. However, when the lookahead completes, KN is reset back to 1, but the relator defining phase is started immediately - whether or not all of the subgroup generators are closed. Therefore, no more information is obtained from these subgroup generators and it is possible that they will be inconsistent with the final coset table.

I changed the program so that when SUBGRP=TRUE and the lookahead phase is entered, the program goes back to the first subgroup generator and scans them all again. I had to put another statement into this loop to stop DEFFLG, (the variable

which determines whether or not we are in the defining phase), being made TRUE at any point if LOOKAH=TRUE. After the lookahead has been applied to all of the subgroup generators, the statement which makes SUBGRP false is skipped over. Then the group relators are processed. When the lookahead completes, we check that SUBGRP=TRUE. If it is, we start the subgroup generation part of the defining phase from scratch, if not, we start the relator defining phase.

(4) In the subroutine PCL12 we find the ultimate irredundant values, HIGH′ and LOW′, of the coincident cosets HIGH and LOW. -HIGH′ is then supposed to be filled in to the first column of each coset which occurs in the HIGH chain and the same is done for LOW′. However, although that is what the program says, it is not actually what it does. Actually, -LOW′ and -HIGH′ are only filled in to the first column of the first coset in their respective chains. I found that two IF statements had been reversed.

The program has two lines :

IF (LOWSH-LOW) 50,60,60 (1)

and IF (HIGHSH-HIGH) 100,110,110 (2)

which should be replaced by

IF (LOW-LOWSH) 50,60,60 (3)

and IF (HIGH-HIGHSH) 100,110,110 (4)

Here LOW is the current irredundant coset in the LOW chain, and LOWS is the initial value of LOW. LOWSH starts off as the second coset in the chain -SPACE(1, LOWS). * Now we set SPACE(1, LOWS)=-LOW and execute the IF statement in (1) above. Obviously, LOWSH≥LOW, so the loop is exited. If (1) is replaced by (3), we go to the statement labelled 50. This sets LOWS=LOWSH and LOWSH=-SPACE(1, LOWS) and the process is repeated from * until LOWSH becomes LOW. The argument is the same for the HIGH chain.

§2.5 Reidemeister-Schreier Presentation

George Havas wrote an implementation of the Reidemeister-Schreier process in 1973 [19]. In this he found the values of the coset representatives and Schreier generators explicitly in terms of the original group generators, using a list structure to deal with the resulting words of variable length. He also had a section to simplify the final presentation - however this was not very efficient.

The implementation which is part of TC2 is very different. Here the final presentation is on an abstract set of Schreier generators i.e. on the generating symbols $s_{K,g}$ rather than on the words $Kg\overline{Kg}^{-1}$. This presentation is then read in to another program, TTRANS, which is used to perform a variety of Tietze transformations - either automatically or manually. It is usually very successful at eliminating redundant generators and further simplifying the presentation. I will describe this program TTRANS in Section 2.6.

The Reidemeister-Schreier process in TC2 is called from subroutine OPT with option RS but, obviously, before this can be done, we need a completed coset table. It does not matter what method of enumeration is used, but before the Reidemeister-Schreier routines themselves are called, the table is compacted to get rid of any redundant rows and to ensure that the coset numbers are in numerical sequence. Then, if the coset table is not standard, a minimal spanning tree is constructed and placed in the buffer above the table, if one does not already exist. Three main subroutines are involved in the Reidemeister-Schreier process itself: COSDEF is used to set up a table of Schreier generators, and RELS and CALWORD then rewrite the group relators in terms of the Schreier generators. The final presentation is written to a file called PRESN.DAT.

Constructing the Minimal Spanning Tree

This sets up a 1-d array above the coset table, which contains a sequence of possible definitions for each coset n>1. For each coset number N, defined as M.COL, where M is another coset and COL is a column of the coset table representing $g^{\varepsilon} \in X^{\pm 1}$, we have :

```
Y(LASTPS+2*(N-1)+1)=M
Y(LASTPS+2*(N-1)+2)=COL
```

If the table is standard, i.e. each row has an entry which is smaller than the row number, we do not have to construct this tree. This is because a sequence of possible coset definitions can be easily found by a backtrack method.

First of all we initialize all of the entries in the minimal spanning tree to zero. Then we take variables NAL=1, KN=1, NEXTL=$\emptyset$, where NAL is the number of cosets in the tree so far, KN is the coset being scanned and NEXTL the number of the previous coset in the tree. We also initialize Y(LASTPS+1)=1 to show coset 1 has been defined.

We find J=SPACE(I, KN) for I=1, ... ,NCOL and then look at Y(LASTPS+2*(J-1)+1) for each J. If this is non-zero, coset J has already been defined, but if it is zero, we make

Y(LASTPS+2*(J-1)+1)=KN

and Y(LASTPS+2*(J-1)+2)=NEXTL .

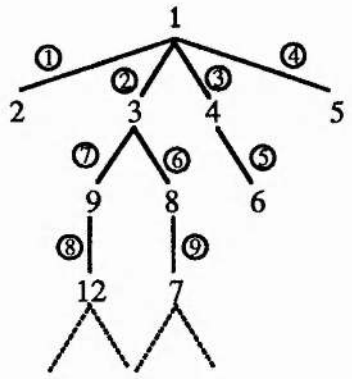
We then update NEXTL to J and add 1 to NAL. When we reach I=NCOL, we check to see if NAL=NALIVE i.e. all cosets have been defined. If not, KN becomes the next known coset already in the tree and NEXTL=$\emptyset$ again. When this row in the coset table has been processed, we look at the next branch in the tree. We keep track of where the next branch and layer is to be found with the variables THISL and NEXTL. Thus, all cosets are defined in terms of those already in the tree, keeping the path to each coset from 1 as short as possible.

E.g.

	1	2	3	4
1	2	3	4	5
2	5	1	8	4
3	1	8	5	9
4	6	5	2	1
5	5	2	1	3
6	8	4	6	6
7	10	12	14	8
8	3	6	7	2
9	9	9	3	12
⋮	⋮	⋮	⋮	⋮

These are the first 9 rows of a larger coset table. It is not standard, since coset 7 has no smaller coset number in any place in its row.

Therefore we have the partial minimal spanning tree :



The cosets are defined in the order shown. Firstly, all primary branches i.e. branches 1-4 above are defined, then all secondary ones (5-7), processing the primary branches in the reverse order of definition. Then, if not all cosets have been defined, we start processing the secondary branches to find tertiary ones, and so on. At this point the locations Y(LASTPS+2*(N-1)+2) in the minimal spanning tree link together the cosets of each layer of the tree, in the order of their insertion into the tree.

←1→	←2→		←3→		←4→		←5→		←6→		←7→		←8→		←9→		←10→		←11→		←12→		...
1	ø	1	ø	1	2	1	3	1	4	4	ø	8	12	3	6	3	8	ø	ø	ø	ø	9	ø	...
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

When all of the cosets have been entered into the tree, we can fill in the column numbers in Y(LASTPS+2*(N-1)+2), overwriting the pointers to the last coset to be defined. This is done for each coset N by looking up M=Y(LASTPS+2*(N-1)+1), then searching row M for the first column, COL, in which N appears. COL is then written into Y(LASTPS+2*(N-1)+2).

The first location of the buffer, Y(BUFFER), is moved to just above the minimal spanning tree.

COSDEF

Here we set up a 1-d array of length NDGEN*NALIVE and for each location I, $1 \leq I \leq$ (NDGEN*NALIVE), we make Y(BUFFER+I)=I*2+100.

The even numbers greater than 102 are our Schreier generator symbols. We now set some of these locations to zero corresponding to the coset definitions. This is done directly from the coset table, if it is standard, or with the aid of the minimal spanning tree, if it is not.

If the coset table is standard we look at the last row, M, initially, and search for the minimum entry, MIN, which lies in column, COL. The equation $M=MIN(COL)^{-1}$ is then taken to be the definition of coset M. If $(COL)^{-1}$ corresponds to a generator, (rather than the inverse of a generator), we make :

$Y(BUFFER+(MIN*NDGEN+(COL)^{-1})-NDGEN)=\emptyset$

otherwise $Y(BUFFER+(M*NDGEN+COL)-NDGEN)=\emptyset$.

Thus, in the first case we make the Schreier generator $s_{MIN,COL^{-1}}=\emptyset$, in the second $s_{M,COL}=\emptyset$ [see Section 1.4]. M is then reduced by 1 and the process repeated. This is continued until M=2 has been similarly processed.

If the table is non-standard, we find MIN for each M from Y(LASTPS+2*(M-1)+1) and COL from Y(LASTPS+2*(M-1)+2). The rest of the procedure follows the standard case.

RELS and CALWORD

We go through the process of tracing out each relator at least twice. Except for the last time, nothing is written to the file PRESN.DAT. However, the table of Schreier generators is altered in each of the other traces. If the traced relator has length 1 we know that this Schreier generator is trivial, so the appropriate location is set to zero in the table. If the relator has length 2, and is not trivial or an involution, the larger of the two Schreier generators is replaced by ±(the smaller), whichever is appropriate. If the table of Schreier generators has been altered in the tracing, all of the relators are retraced to check that no more Schreier generators are trivial or equal to another.

When the tracing eventually results in no more changes to the table, the next trace does write the relators to PRESN.DAT. These are first written into the buffer above the Schreier generator table, all zero entries being discarded. Then, if the relator has length > $\emptyset$, it is written to PRESN.DAT in the same form as those subgroup relations obtained from SUBGROUP i.e. it is preceded by the exponent 1 and

followed by 99. If we do find a trivial relator, $\emptyset$ is written to the next line of PRESN.DAT . This has to be done, because we say in the first line of PRESN.DAT that there are NALIVE*NDREL relations in the file.

It is also possible to rewrite the subgroup generators if we call RS with a negative parameter, although the resulting relators are not necessary to the final presentation. In this case each relator, obtained by tracing out the word, is preceded by minus the number of the subgroup generator. Also, the number of subgroup generators, NSUBGP is added to the number of relations in the first line of PRESN.DAT . (Note that a maximum of 101 subgroup generators can be rewritten, since the Schreier generators start at 102.)

By the construction of the coset representatives it is obvious that they form a Schreier system, since each coset, I, and hence each coset representative, $\overline{I}$, is defined in terms of one already defined.

Comparing our presentation with that in Theorem 6 on page 22, we have certainly found the set of relations

$$\tau(K R_j K^{-1}) = 1.$$

As for the first set of relations, $s_{M, g_{i_r}} = 1$ where $M g_{i_r} \approx \overline{M g_{i_r}}$, we must show that these Schreier generators have already been omitted from the presentation, i.e. were set to zero in the Schreier generator table.

If $M g_{i_r} \approx \overline{M g_{i_r}}$, then $M g_{i_r} \overline{M g_{i_r}}^{-1} \approx 1$.

Thus, if $\overline{M g_{i_r}} = m_1 m_2 \ldots m_n$, where $m_1 m_2 \ldots m_n$ is freely reduced then

$$M g_{i_r} m_n^{-1} \ldots m_2^{-1} m_1^{-1} \approx 1 .$$

Let $M = a_1 a_2 \ldots a_p$ where $a_1 a_2 \ldots a_p$ is freely reduced. i.e.

$$a_1 a_2 \ldots a_p g_{i_r} m_n^{-1} \ldots m_2^{-1} m_1^{-1} \approx 1 .$$

Thus g_{i_r} either cancels with a_p or m_n^{-1},

i.e. $a_p = g_{i_r}^{-1}$ or $m_n = g_{i_r}$.

(i) $\underline{a_p = g_{i_r}^{-1}}$

Cancelling this pair we obtain :

$$a_1 a_2 \ldots a_{p-1} m_n^{-1} \ldots m_2^{-1} m_1^{-1} \approx 1 .$$

Since we cannot cyclically reduce the left hand side, and $a_1 a_2 \ldots a_p$ and $m_1 m_2 \ldots m_n$ are freely reduced, a_{p-1} must now cancel with m_n^{-1}. Then, a_{p-2} cancels with m_{n-1}^{-1} and so on. Thus,

$$a_1 = m_1,\ a_2 = m_2,\ \ldots\ldots,\ a_{p-1} = m_n .$$

So, $M g_{i_r}$ is exactly the same word as $\overline{M g_{i_r}}$.

(ii) $m_n = g_{i_r}$

This works in exactly the same way and we find that

$$a_1 = m_1,\ a_2 = m_2,\ \ldots\ldots,\ a_p = m_{n-1} .$$

Thus, in both cases, $\overline{M g_{i_r}}$ is defined as $M g_{i_r}$ if $M g_{i_r} \approx \overline{M g_{i_r}}$. Hence, the relations $s_{M, g_{i_r}} = 1$ in presentation (20) of Chapter 1 are simply those Schreier generators obtained from the definitions of each coset representative. Their symbols were not incorporated into the relations in PRESN.DAT, so we do not have to include these relations explicitly in our final presentation.

§2.6 A Tietze Transformation Program - TTRANS

The first version of this program to carry out Tietze transformations was written in the early part of 1977 by P. Kenne at ANU, Canberra. It was originally designed to improve the simplification stage of G. Havas's Reidemeister-Schreier program described in [19]. Two revised versions of the program by J.S. Richardson appeared later that year, then, in 1981, E.F. Robertson of St. Andrews modified it further to simplify the output from the modified Todd-Coxeter program, SUBGROUP [2]. This latter version is described in [20] and has been used successfully in many different areas of Group Theory. For example, this program and its earlier versions have been used to find presentations for simple groups [12], [11]; to investigate Fibonacci groups [21] and to determine non-abelian tensor products of groups [6].

One later amendment was made in 1986, again by E.F. Robertson, when a "weighted substring search" facility was added. A description of this, and some

results which have been obtained from it, appear in [32]. I will explain "weighted substring searching" in Section 2.6.1.

This latter version of TTRANS is the one which I have adapted to simplify the output obtained from my program MODTC. I have made a few changes to the existing code and also added some new facilities. This new version, NTTRANS, is discussed in Section 3.3, but first I will describe the implementation of TTRANS itself, giving an outline of how the program works, rather than investigating the actual code in detail.

The program requires a set of relators and a decoding tree, TREE.DAT, as input. The latter is generated by SUBGROUP, but if the relators in question were not obtained from the modified Todd-Coxeter algorithm, any such file will suffice (as long as we do not try to decode any generators from the tree!). The relators themselves can be input from a file, or input one at a time from the keyboard. All relators are preceded by the exponent and followed by 99, marking the end of the relator.

As they are read in, the relators are sorted into ascending canonical order, based on the normal integer ordering of the generators.

If x and y are generators, we define the ordering $<$ on the relators by :

$x^{\varepsilon} < y^{\delta}$ if $x < y$ or $x=y$ and $\varepsilon = 1, \delta = -1$

and $x_1^{\varepsilon_1} x_2^{\varepsilon_2} \ldots x_m^{\varepsilon_m} < y_1^{\delta_1} y_2^{\delta_2} \ldots y_n^{\delta_n}$ if $m < n$.

Hence, inductively,

$$x_1^{\varepsilon_1} x_2^{\varepsilon_2} \ldots x_m^{\varepsilon_m} < y_1^{\delta_1} y_2^{\delta_2} \ldots y_m^{\delta_m} \quad \text{if} \quad x_1^{\varepsilon_1} x_2^{\varepsilon_2} \ldots x_{m-1}^{\varepsilon_{m-1}} < y_1^{\delta_1} y_2^{\delta_2} \ldots y_{m-1}^{\delta_{m-1}}$$
$$\text{or} \quad x_1^{\varepsilon_1} x_2^{\varepsilon_2} \ldots x_{m-1}^{\varepsilon_{m-1}} = y_1^{\delta_1} y_2^{\delta_2} \ldots y_{m-1}^{\delta_{m-1}} \quad \text{and} \quad x_m^{\varepsilon_m} < y_m^{\varepsilon_m}.$$

As the canonical representative of each relator we choose the smallest, with respect to this ordering, of the set of relators made up of all cyclic permutations of the relator and its inverse.

The relators are added one at a time so that it is easy to spot any relator which is a duplicate of one already in the list, since they will have identical canonical forms. If this happens, the new relator is discarded.

The relators are stored in a large array, GEN. Before any relators are read in, the locations in GEN are linked together in groups of three, the first word of each

"node" being linked to the first word of the next. This is basically a linked list of free space and the variable FREEG keeps track of the first free word. As relators are read in, this list of free space is gradually shortened node by node. Any space later freed by Tietze transformations shortening relators is relinked back in to the list.

The list holding the order of the relators is kept in the array RELLST in GEN. The variable TOP points to the first relator in the list and BTM to the last relator. Each relator has a node of three words containing the following information :

...	pointer to information associated with that particular relator in array LELST	pointer to next relation or ∅ if last	'name' of relator	pointer to previous relator, ∅ if first	...

The 'name' of the relator is an integer identifier which is kept with that relator throughout the execution of the program. This allows you to be able to tell which of the original relators are left in the final presentation.

The array LELST again has one node for each relator. These look like this :

...	length of relator	the exponent (if > 1) otherwise the value of MINPOINTER	pointer to first letter of the relator	...

If the exponent is greater than 1, the first word holds the length of the unexponentiated word, not the total length of the relator. If the exponent is 1, the second word holds the MINPOINTER. This is used to speed up substring searching and is a negatively flagged pointer to an occurrence of the least frequently occurring generator in the presentation, if it exists in that relator. If it does not exist it points instead to the last letter of the relator (not the penultimate as is indicated in the program's comment statements).

The relators themselves are stored as circular doubly-linked lists. This enables any cyclic permutation of a relator or its inverse to be found quickly. Each letter in each relator occupies one of the 3-word nodes as follows :

...	generator	pointer to previous node	pointer to next node	...

When a relator is used to eliminate a generator from the presentation, it is transferred to yet another list - the list of elimination rules. This has the same format as RELLST, but this time the top and the bottom are pointed to by ERTOP and ERBTM. This list is useful for reconstructing the order in which eliminations have been carried out.

There is yet another list stored in GEN which keeps track of the number of times each generator occurs in the list of relators. This is useful because it is often a good idea to eliminate generators which occur least frequently in the presentation, as this does not usually increase the overall size too much.

The main program of TTRANS is similar to the subroutine OPT in TC2, where commands are input as a combination of one or two letters followed by an integer argument, ARG. In the next few pages I will describe the main functions of the program and the commands used to control these.

The program has four main facilities for performing Tietze transformations :

(i) Elimination of redundant generators.

(ii) Substring searching (i.e. looking for long substrings which can be replaced by equivalent shorter strings).

(iii) Equal length string substitutions.

(iv) Short string replacement.

The first three of these, together with certain tree-decoding commands, can be used to simplify presentations found with the modified Todd-Coxeter algorithm. There are also "weighted substring" versions of (i) and (ii).

(i) Elimination of Redundant Generators

There are different strategies available for choosing the order in which generators are to be eliminated.

A certain generator can be specified by the user. For example, EG14 will

eliminate generator 14, assuming that it occurs only once in some relator. The relators are checked in ascending canonical order to find the first one fitting this criterion. Thus, the chosen generator will be replaced by the shortest possible string. If no generator is specified, i.e. ARG=$\emptyset$, the highest numbered generator in the first relator which contains a single occurrence of a generator will be eliminated.

It is also possible to perform eliminations using a specified relator. For example, EL2 will use relator 2 to eliminate the highest numbered generator which occurs only once in the second relator.

Yet another command, ES, can also be used to eliminate redundant generators. This performs "short eliminations", that is, all relators of length one or two are used to eliminate generators.

Once a suitable substring to replace the chosen generator, ELT, has been found, we replace ELT by this substring each time it occurs in each of the relators. After the relator has been altered, it is freely and cyclically reduced, its new canonical form found, and if necessary it is moved to a new position in RELLST.

If ELT is trivial or equal to another generator we start searching for occurrences of ELT in the relators starting at TOP, BTM otherwise. This course of action is taken to try to prevent reordered relators from being searched twice.

In the first two cases, eliminating ELT from a relator cannot make it longer - it will either shorten it or the relator will remain the same length. Thus, it is not very likely that it will be moved down the list when it is reordered, so will not be searched again for ELT. [It can only be moved down if a lower-numbered generator was replaced by a higher numbered one.]

Similarly, in the last case, eliminating a generator using a relator with length greater than two, will usually lengthen the other relators. Thus, if we start searching for ELT in the longest relators first, we are unlikely to have to process that relator again once it has been reordered. [It is, of course, possible for the new relator to be freely and cyclically reduced, so that it ends up shorter and is moved up RELLST.]

(ii) Substring Searching

Each relator in turn is taken to be the current test relator LAUREL. Then each relator after LAUREL in RELLST is searched for a substring common to LAUREL, which

is over half LAUREL's length. When a match is found, this other relator, HARDY, is shortened by substitution.

This is done in the following way :

We write LAUREL into a 1-d array, LRL. This array has only 1500 locations, so any relator with length ≥ 1500 is passed over. We write LAUREL into LRL starting with the first letter of LAUREL if its exponent is > 1, or the first letter after the MINPOINTER if the exponent is 1. The first location of LRL is always left empty, and if the exponent is greater than 1, LAUREL is expanded fully into LRL. Then, if LAUREL has even length, L, LRL(1) is set equal to LRL(L+1). This ensures that we always have a middle generator in LRL and as long as we only look for matches with length ≤ L, it is consistent. The whole idea of keeping the generator pointed to by MINPOINTER to the end of LRL is that, since it is the rarest generator in the presentation, it is the one with which you would be least likely to find a match in HARDY.

ELT then starts equal to this middle generator. We search each relator, HARDY, in turn for ELT or its inverse. If we find an occurrence we scan backwards, then forwards, until no more generators correspond. If the match is too short we find the next occurrence of ELT and try again. The first match found which is greater than half the length of LAUREL is used for the substitution. We then check for any more matches in HARDY, before updating HARDY to the next relator. When each relator in turn has been HARDY we have finished with this value of LAUREL, if the exponent of LAUREL is greater than 1, otherwise we have to cyclically permute the relator in LRL by [L/2]+1 places, and repeat the whole process. Thus, we search each relator HARDY for two diametrically opposite generators in LAUREL. Obviously, any substring of LAUREL with length > L/2 contains at least one of these two generators, so if any matches in HARDY of a suitable length exist, we will always find one. This does not need to be done if the exponent is greater than 1 because any substring of HARDY which has length > L/2 must contain the original generator, ELT. Note that if HARDY has exponent > 1, we only look for substrings in the unexponentiated subword and not in the expanded relator.

When each relator with expanded length ≤ 1499 has taken its turn as LAUREL, we have completed one pass of the substring search. Often the shortened relators will

give further matches, so usually more than one pass is made. It is possible to assign a value between $\emptyset$ and 1 to the real variable RECOVR. Then RECOVR*OLDSIZ is calculated to see if the length of the presentation has been reduced by a certain percentage. If it has, another pass is made, otherwise nothing more is done.

Again there are several commands to control substring searching. The procedure outlined above is carried out with the command SF. To perform substring searching with only one specified relator, e.g. relator 3, as LAUREL, the command SU3 is input.

At the end of the substring search routine we attempt to simplify the presentation further by using commutator relations (if they exist) to bring occurrences of the same generator together.

(iii) Equal length String Substitutions

Here, if we have a relator w_1w_2 with the length of w_1 equal to the length of w_2, it is often worthwhile replacing all occurrences of w_1 by w_2, or vice versa. Then, when a substring search is done, the presentation can often be shortened further. These equal length substitutions are carried out with the IV command which can have a range of argument values to control the type of equal length substitution done.

ARG = $\emptyset$, 1 use all relators except involutory generators. If $w_1w_2=1$ we search for a negative match in HARDY, i.e. seek to replace w_1^{-1} by w_2 or w_2^{-1} by w_1.

ARG = -1 inverse of ARG=$\emptyset$ substitutions so looks for a positive match in HARDY i.e. seeks to replace w_1 by w_2^{-1} or w_2 by w_1^{-1}.

ARG = 2 replace inverses of involutory generators by their inverses, i.e. if $a^2 = 1$ replace a^{-1} by a everywhere.

ARG = -2 inverse of ARG=2 substitutions so replace involutory generators by their inverses, i.e. a by a^{-1}.

ARG = 3, -3 make random substitutions of equal length.

ARG = 4 replace the equal length substring with the largest generator sum by the one with the minimum generator sum.

Apart from ARG=±2, these are carried out in a similar manner to the substring searching and replacement described in (ii). For ARG=∅, 1 we search only for occurrences of -ELT in HARDY, and for ARG=-1 we search only for occurrences of +ELT. When ARG=±3 a random mixture of ARG=1 and ARG=-1 equal length substitutions are carried out, a random number generator being used to decide the next value of ARG after each individual substitution has been made. If ARG=4 we look for equal length matches of both the ARG=1 and ARG=-1 types. When we find one we calculate the sum of the absolute values of the generators in the matching substring in HARDY. If this is smaller or equal to the total sum for LAUREL we do nothing, otherwise we carry out the substitution.

(iv) Short String Substitutions

These are carried out via the SS command, and a new generator, g_i, is substituted for the string of length two, $g_p^{\varepsilon_p} g_q^{\varepsilon_q}$, which occurs most often in the presentation. This new generator is numbered one higher than the highest numbered existing generator. Then, if ARG=∅ we eliminate the generator which now occurs least often of g_i, g_p and g_q. If ARG=1 we eliminate the first of the generators in the original string of length two, if ARG=2 we eliminate the second. There is yet another possible argument, 3, which carries out a sequence of short string replacements : SS1 followed by SS, SS2 and finally SS again.

There are several other general commands which are useful for simplifying the presentation further or tidying it up. These are :

FX This finds exponents, if they exist, for all relators. E.g. if a relator is 1 2 1 2 1 2 it will be rewritten as (3) 1 2 where 3 is the exponent.

RP replaces two relators which are both powers of the same word by one relator whose exponent is the h.c.f. of the two powers.

FC finds commutator relations (if any exist) and uses them to bring occurrences of the same generator together.

There are two main tree-decoding commands for use with output from the modified algorithm program, SUBGROUP. These are EH, standing for "eliminate high numbered generator", and DC , meaning "decode".

EH Here generators are eliminated starting with the highest number first. The existing subgroup relators are used for this if possible, otherwise the relator from the tree decoding this generator is added to the list of relators and this used to eliminate the generator. This process may add one or two high numbered generators not previously present in the presentation, which, in turn, may have to be eliminated from the tree. After ARG eliminations a substring search, SF, is done. If ARG<$\emptyset$ we stop the elimination without going to the tree, if we cannot eliminate the generator from the other relators.

DC This decodes generator ARG. If ARG<$\emptyset$ all generators greater than or equal to BIGGEN are eliminated, starting with the highest numbered. If ARG=-999 only the tree relations are used for eliminations. If ARG≠-999 the shortest possible relation, after the tree relation has been added, is used. IABS(ARG) is the number of eliminations done between substring searches, with no searches carried out if ARG=$\emptyset$. Note that BIGGEN is the smallest high-numbered generator to be decoded, and is the value of GENEND in SUBGROUP, the number of relators from the Reidemeister Schreier routine in TC2, or 100 otherwise.

§2.6.1 Weighted Substring Searching

The substring searching routine described earlier always tries to minimize the free length of the relators, i.e. the length of the words in the free group. However, it is not always the case that shortening the relators at every stage will lead to the shortest possible free length in the final presentation. Often, in hand calculations, a relator is lengthened using other relators until it suddenly collapses, allowing a short relator to be deduced. One method of allowing this to happen is to carry out "weighted substring searching". In this, each generator can be allocated its own weight. Then, when the substring searching is carried out, the combined weight of the generators in LAUREL (w_1w_2=1) is calculated and only substrings w_1 in HARDY which have a higher combined weight than w_2 are replaced. Thus, the generators which we would prefer to be present in the final presentation should be given the lowest weights.

The substring searching itself is slightly different to that already described. To start with LAUREL is copied twice into LRL. (Thus, since LRL still has size 1500 we can only deal with relators LAUREL which have 750 letters or fewer.) ELT now

becomes the first letter of the second copy of LAUREL, and we ensure that each match we find has length less than, or equal to, the length of LAUREL. Like the ordinary substring searching routine, we now repeat the process with another generator, ELT, if the exponent is 1. This time, instead of the new value of ELT being chosen so that it is halfway round the relator from its old value, we choose it so that it is halfway round with respect to the combined weights of the generators; that is, if the relator r is $g_1^{\varepsilon_1} g_2^{\varepsilon_2} \ldots g_k^{\varepsilon_k}$, the first value of ELT will be $g_1^{\varepsilon_1}$ and we choose the next value to be $g_t^{\varepsilon_t}$ where

$$\text{weighted length}\,(\, g_1^{\varepsilon_1} g_2^{\varepsilon_2} \ldots g_t^{\varepsilon_t}\,) \;\geq\; \tfrac{1}{2}\,\text{weighted length}\,(\,r\,),$$

$$\text{weighted length}\,(\, g_1^{\varepsilon_1} g_2^{\varepsilon_2} \ldots g_{t-1}^{\varepsilon_{t-1}}\,) \;<\; \tfrac{1}{2}\,\text{weighted length}\,(\,r\,).$$

The command used to carry out this weighted substring search is WS. By default, at the beginning of the program a 1-d array of weights is set up, with the weight of each generator set to the number of the generator. This can be changed. If WS is called with ARG<∅, all of the weights are set to 10, and then the user is asked which generators he wishes to change, and the new weights to be allocated. Then, the weighted substring search is carried out as before.

From this it is obvious that weighting all generators equally will give us the usual method of substring searching. However, because of the different arrangement of LAUREL in LRL, some of the matches found could be very different.

E.g. LAUREL=$a^{10}bab$ HARDY=$a^{11}ba^2b$

If one was attempting this by hand, the obvious substring to substitute for in HARDY is $a^{10}ba$, since this gives the longest match with LAUREL. However, this is not done with either SF or WS.

With SF, assuming neither a nor b is the rarest generator in the presentation, the first letter of LAUREL is written into the second location in LRL, the second in the third location and so on. LRL(1) is left empty, since LAUREL has an odd number of letters. ELT then becomes the middle generator, the 7th.

	a	a	a	a	a	a	a	a	a	a	b	a	b
							↑						
							ELT						

Now HARDY is scanned from left to right, looking for a match with ELT.

1st a in HARDY gives us a match of a^4..... too short.

2nd a in HARDY gives us a match of a^5..... too short.

3rd a in HARDY gives us a match of a^6..... too short.

4th a in HARDY gives us a match of a^7.

Now we have a match greater than half the length of LAUREL, so we replace a^7 in HARDY by $a^{-3}b^{-1}a^{-1}b^{-1}$.

However, with WS we have the following array LRL :

a	a	a	a	a	a	a	a	a	a	b	a	b	a	a	a	a	a	a	a	a	a	a	b	a	b
													↑												
													ELT												

Now, matching ELT with the first "a" in HARDY, we have a match of length 12, aba^{10}, so aba^{10} in HARDY is replaced by b^{-1}.

Thus, we can get better answers because we are now "wrapping round" LAUREL which we did not do before. However, we still do not necessarily get the longest possible match. To do that, the length of each match would have to be found for a certain relator HARDY, and the longest one used for the substitution. This would take a long time, especially if the presentation was lengthy.

There is also a weighted substring generator elimination facility. This is carried out with the command WE and replaces generators by strings of lower weight. This is done by calculating the weight of a relator, IWT, then searching the relator to see if it has a generator which alone has weight greater than IWT/2. This generator is then eliminated from the presentation.

I actually found an error in this process. The wrong parameter was passed to the elimination routine which meant that the highest numbered generator occurring in the chosen relator was eliminated, <u>not</u> the generator with the largest weight. The two do not necessarily tally since the default weighting can be altered by hand. I rectified this, so that the generator in the relator with the largest weight greater than IWT/2 is found and eliminated.

There is also a weighted substring version of EH, called by WH. This is identical to EH, except that weighted substring searching is carried out every ARG eliminations, instead of the ordinary substring searching.

§2.6.2 Automatic Simplification Commands

There are a number of automatic simplification facilities included in the program – AU, WA and GO.

AU The argument here is the number of generators to be eliminated between substring searches, SF. However, if we eliminate a generator, replacing it by a substring with length ≥ 5, the counter keeping track of the number of eliminations done since the last substring search is incremented by an amount depending on the length of the substitution. This ensures that the presentation will not become too long before the next substring search is carried out. The user is asked to input a parameter to control the type of eliminations done. The possible values are :

- $\emptyset$ to perform short eliminations. This uses the shortest relators first to try and eliminate any generators which occur only once in a relator.
- 1 to eliminate the generators in order of increasing frequency in the presentation, i.e. the rarest generator is eliminated first.
- 1$\emptyset$ this is for strategy 1 followed by strategy $\emptyset$ i.e. all possible eliminations of type 1 are carried out then eliminations of type $\emptyset$.

After all possible generators are eliminated, we then carry out the following sequence of commands :

FX, IV-2, SF, IV-1, SF, IV, SF, IV, SF, IV2, SF,
IV3, SF, IV3, SF, IV3, SF, IV3, SF, IV3, SF.

WA This is identical to AU , except that all substring searching is weighted.

GO This command is used to eliminate as many generators as possible with numbers greater than, or equal to, BIGGEN. First of all it searches for relators containing one high-numbered generator i.e. only one generator greater than, or equal to, BIGGEN. When it finds one, it eliminates this generator. Then it looks for all relators with only two high-numbered generators and eliminates the largest. This is then done for relators containing 3, 4 and 5 high-numbered

generators in turn. A substring search is carried out after each elimination and the process started again, searching for relators with single high-numbered generators.

§2.6.3 Other Useful Facilities

There are some other useful facilities in TTRANS worth a mention :

The command TY-1 tidies the generator numbering. It numbers the generators sequentially from 1 again, with the value of TDYGEN being the number of generators in the presentation.

In many of the elimination routines you are asked to specify the number of a checkpoint file. The presentation is written to this periodically. Thus, the execution can be halted at any point and the simplification restarted from the checkpoint file at a later date. This is especially useful on systems where you are allocated a certain amount of CPU time. In that situation, if a simplification takes much longer than expected and you run out of CPU time, you do not have to start again from scratch.

When a presentation is written to a file it is arranged in the following format :

```
IDENT   WIDTH   TDYGEN   NUMREL
EXPONENT 1                RELATOR 1      99
EXPONENT 2                RELATOR 2      99
    ⋮                         ⋮           ⋮
EXPONENT N                RELATOR N      99
          1                       997
EXPONENT 1     ELIMINATION RULE 1        99
EXPONENT 2     ELIMINATION RULE 2        99
    ⋮                         ⋮           ⋮
EXPONENT M     ELIMINATION RULE M        99
          1                       999
```

Here, IDENT is the group name (if one has been allocated); WIDTH is the field size of each integer; TDYGEN is the number of generators if the generators have been tidied, otherwise zero; and NUMREL is the number of relators.

Checkpoint files can be taken at any time from the main program with CP(ARG) and read in with the RS(ARG) command.

To read in relators from a file we can also use the command NF. This gives us the option of reading in selected relators only. NF-1 reads in all of the relators from PRESN.DAT. To read from another file, we can change the input file to FOR015.DAT, say, by the command NF-15. Then, NF12 will read in the first 12 relators from FOR015.DAT, whereas the command NS4 will skip four relators.

It is also possible to change to "word mode". This suppresses cyclic reduction, rewriting in canonical form and reordering relators. Certain processes cannot be performed in this mode, e.g. substring searching or fixing commutators.

There are various other commands, which I will not list here, for writing out the relators and listing the group generators, together with their total number of occurrences.

Chapter 3

New Implementations of the Computer Programs

This chapter describes various computer programs which are based on the programs discussed in Chapter 2. Again I will refer to the versions running on the VAX 11/785.

§3.1 An Adaption of the Program SUBGROUP - NEWREL

When carrying out the modified Todd-Coxeter algorithm as implemented in SUBGROUP, as soon as coincidences occur there are possible extra subgroup relators which can be collected. These are redundant as far as the final presentation is concerned. However, in many cases, they are shorter than many of the final relators obtained from SUBGROUP and, by adding them to the presentation, the presentation can be quickly reduced, both in terms of the number of relators and their combined length. Sometimes, for example, these "extra relators" can give the period of an element which cannot be obtained very easily from the other relators.

This idea of picking up "extra relators" came from a paper by M.J. Beetham and C.M. Campbell [4]. In this, an example was worked through by hand with the modified algorithm and all possible information extracted from coincidences. Their resulting presentation had four relatively short relations, whereas only three longer ones would have been found using the traditional modified algorithm. I will explore this example later in Chapter 4. Basically, all of these "extra relators" are obtained from coincidences which yield no new information. These coincidences are of two types :

(i) A relator closes under a scan with coset α or a subgroup generator closes with coset 1, yielding no new information. Here, we actually have a coincidence between the cosets IFRONT and IBACK.

(ii) In the coincidence routine coset α is found to be coincident to coset β but, on tracing through the linked lists in the second column of SPACE, both are found to be coincident to a third coset γ.

I will look at these in more detail :

(i) This can happen either in the defining or the lookahead phases. It may be that some or all of the cosets encountered in this scan will eventually disappear with subsequent coincidences, and the corresponding h-words lengthened due to concatenation with coincidence words. Thus, when the final relator is traced out at the end of the process, it is very much longer than this "intermediate relator". The information from this "intermediate relator" is not lost but is incorporated into other longer relators and often cannot be extracted easily. I altered SUBGROUP so that any intermediate relator found in this way is written to a file, FOR030.DAT, as a single high-numbered generator built up from the concatenation of the h-words in the word table when tracing out this row in the relation table. These relators are not written directly to PRESN.DAT since their addition can greatly increase the size of the final presentation in many cases. Often we only wish to add two or three short extra relations to help simplify those in PRESN.DAT .

(ii) In this case we obtain the equation ICOINWR.$\beta = \alpha$ with $\beta > \alpha$. However, on entering COINC, we find that ICOINWR$_\beta$.$\beta = \gamma$ and ICOINWR$_\alpha$.$\alpha = \gamma$, where ICOINWR$_\beta$ and ICOINWR$_\alpha$ are obtained by concatenating the h-words in the first column of all of the cosets in the coincidence chain between the pairs of cosets β, γ and α, γ respectively.

$$\begin{aligned} \therefore \quad & \text{ICOINWR}_\alpha \, \text{ICOINWR}.\beta = \text{ICOINWR}_\alpha.\alpha \\ \Rightarrow \quad & \text{ICOINWR}_\alpha \, \text{ICOINWR}.\beta = \gamma \\ \Rightarrow \quad & \text{ICOINWR}_\alpha \, \text{ICOINWR} \, (\text{ICOINWR}_\beta)^{-1}.\gamma = \gamma \\ \Rightarrow \quad & \text{ICOINWR}_\alpha \, \text{ICOINWR} \, (\text{ICOINWR}_\beta)^{-1} = 1 \end{aligned}$$

In my adaption of SUBGROUP this new relation is written to the file FOR031.DAT, again in the form of a single high-numbered generator.

In order to implement this version of SUBGROUP, which I shall call NEWREL, several changes had to be made to the program. First of all the size of the tree, and its corresponding file TREE.DAT, had to be increased in size considerably. This is because SUBGROUP overwrites parts of the tree at various points in the program where h-words are concatenated but nothing is written into the word table.

This happens at precisely those places where we are now collecting the extra relations, so these parts of the tree are needed to decode those relations written to FOR030.DAT and FOR031.DAT and cannot be erased. Strangely enough, no part of the tree is overwritten if a relator scan fails to close in the lookahead phase. I would have thought that this situation would have introduced many unnecessary entries into the tree. Thus, the final presentations in the files PRESN.DAT obtained from the two programs, SUBGROUP and NEWREL, will each have a different set of high-numbered generators. However, these presentations are in fact the same, and will be decoded back to the same presentation in the original subgroup generators.

As well as suppressing the overwriting of the tree if a subgroup generator is found to be redundant (i.e. the subgroup generator closes during a scan with coset 1 with no new cosets requiring definition), the line which resets the number of the next subgroup generator to the number of the redundant one has also been omitted. This ensures that, when the relators are collected from the subgroup generators at the end, each generator is allocated the correct number.

I will now outline the main changes I have made to the code of SUBGROUP to enable me to collect these extra relators.

Two new counters, I30 and I31, are initialized to zero at the start of the program. These will keep track of the number of extra relators written to FOR030.DAT and FOR031.DAT respectively, and their values are printed out with messages to that effect when the enumeration terminates.

The relators written to FOR030.DAT and FOR031.DAT are written in the same format as those in PRESN.DAT, prefixed by the exponent 1 and followed by 99.

In the subroutine ENUM, when a relation or subgroup generator closes, either on the forward or backward scans, we arrive at a statement labelled 210 which checks that IFRONT≠IBACK before calling the coincidence routine. In NEWREL, if IFRONT=IBACK we have an extra relator. We then check that the word in the h-words obtained by tracing out the relator/subgroup generator is not trivial i.e. JOIN≠1. If it is trivial we carry on as before, if not, we write our new relator JOIN to FOR030.DAT and increment the counter I30 by 1.

The next amendment is in the subroutine COINC. Before we record a coincidence JA≡JB obtained from comparing two entries in the rows KA and KB in SUBGROUP, we make sure that JA≠JB. In NEWREL if JA=JB we write the coincidence word to FOR031.DAT and update I31 (if the coincidence word ≠ 1).

One other change that I have made is to alter the h-word obtained in ENUM when the forward scan completes. At this point SUBGROUP has the statement :

JOIN = LOW.

It then tests whether the coincidence is trivial and acts accordingly. If this is a non-trivial coincidence occurring on the forward scan of a subgroup generator, the resulting extra relator is wrong because the coincidence word, ±JOIN, is wrong. This happens because the number of that subgroup generator has not been taken into consideration. Thus, JOIN=LOW should be replaced by the line :

CALL CATCON(-HIGH, LOW)

which returns the result of the concatenation to the variable JOIN. This statement is also valid for the completion of the forward scan of a relator because in that case HIGH will just be 1 so JOIN will have the value LOW as before.

After testing my program NEWREL with a range of examples, I found that in many cases I obtained a better final presentation than that resulting from PRESN.DAT alone.

Although there are no set criteria for deciding whether one presentation is better than another, usually one normally wishes to find a presentation with as few, and as short, relations as possible. It is rare to find an example where the presentation with the fewest relations is also the one with the shortest total length, so usually a compromise has to be struck. Often one is also looking for relations which express the period of short words such as the generators, products of pairs of generators, or commutators.

§3.2 Adaption of the program TC2 - MODTC

Impressed by the results from NEWREL, I decided to incorporate the extra relation facility into my new modified algorithm program, MODTC. This is basically the same enumeration program as TC2, (see Section 2.4), but has the modified Todd Coxeter algorithm grafted on to it. This has been done in such a way that both the ordinary Todd-Coxeter algorithm and the modified Todd-Coxeter algorithm can be run with the same package. This is very useful because you can investigate the best way of enumerating the example (e.g. find the method and the associated parameters which define fewest cosets) before the modified algorithm is tackled. As well as HLT, the method available in SUBGROUP, this package allows you to run the modified algorithm under the Felsch strategy. This intuitively seems a better way of attacking problems with the modified algorithm since fewer cosets usually need to be defined. This avoids a lot of coincidences with the associated build-up of long h-words in the word table, hopefully resulting in a better final presentation. The results from the two strategies, Felsch and HLT with Lookahead, are compared in Chapter 4.

First of all I took the coset enumeration program TC2 and converted it to run the modified algorithm, and also to collect the extra relations as outlined in Section 3.1. [At this point the ordinary Todd-Coxeter algorithm could not be run as well.] I then identified all those subroutines which were not essential to the basic algorithm, (for example the subroutine which allows you to add new relators), and I discarded them. The rest of the subroutines were examined very carefully to ensure that they were consistent with the variables and the data structures required for the modified algorithm. This required many changes to the values of the pointers, some of which had to be altered throughout the program. This version was called, for obvious reasons, SHORT.

When I eventually had SHORT working satisfactorily I gradually added back the missing subroutines from TC2, ensuring that they were either adapted to carry out the modified algorithm as well as the ordinary Todd-Coxeter, or that two subroutines were added - one to deal with each algorithm. I decided that I would not adapt the main coset enumeration routines ENUMTL, ENUMFL, APPLY, COINC and PCL12 to do both jobs, as this would make them very difficult to read since they would have to contain many statements prefixed by :

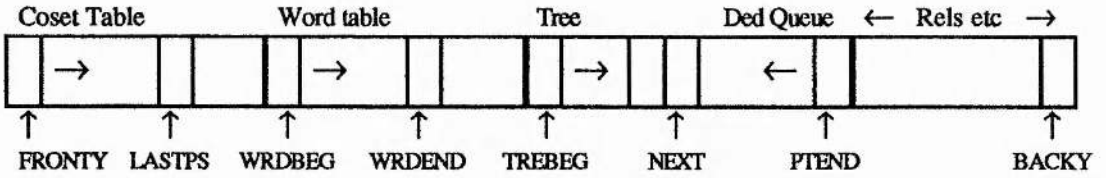
IF (MTC) ... .

Therefore, the versions of these subroutines in SHORT were renamed as MENMTL, MENMFL, MAPPLY, MCOINC and MPCL12, and the original versions added from TC2. This combined version is called MODTC.

The program MODTC evolved over a period of nearly two years. Various major changes were made to it during that time, but I will not describe any which do not exist in the final version.

The data structures for the modified algorithm were carefully chosen to fit in with those for the ordinary Todd-Coxeter process. This ensures that it is easy to switch from one process to the other with very little alteration to the values of the variables.

I decided to put both the tree and the word table in the large array Y. I originally thought of having a variable-sized tree, growing down to reach the word table. However, although this was possible for the HLT method, it did not allow space for the deduction queue in Felsch. Thus, I realised that a bound on the maximum size of the coset table, the word table and the tree had to be entered or calculated in advance. I decided to place the word table after the coset table, as in SUBGROUP, then have the tree reaching up from the end of the word table towards PTEND.



I decided that the maximum size of the tree should be determined by the variable TREESZ. A default value for this is read in from BLOCK DATA but, like most variables, can also be changed from within OPT. This value is subtracted from PTEND+1 to find the beginning of the tree, TREBEG. If we are going to carry out a Felsch enumeration 2 extra locations are left above the tree to hold the beginning of the deduction queue.

The rest of Y is then split up to hold the coset table and the word table. The coset table requires NCOL columns, so the word table needs NCOL+1, the extra one being used to hold the coincidence words. The word table and the coset table always

grow (and collapse) together. Therefore I decided to make :

NMAX = (PTEND-FRONTY-TREESZ-1) / (NCOL*2+1) if Felsch,
NMAX = (PTEND-FRONTY-TREESZ+1) / (NCOL*2+1) if HLT.

Then,

WRDBEG = NMAX*NCOL+FRONTY and
TREBEG = PTEND-TREESZ-1 if Felsch
or TREBEG = PTEND-TREESZ+1 if HLT.

I decided that this method was best since PTEND still references the top of the deduction queue and FRONTY is still the first location in the coset table. Therefore, if we are not carrying out the modified algorithm, all we have to do is to redefine NMAX. This arrangement also allows you to make the coset and word tables larger by making the tree smaller, and vice versa, depending on the storage requirements of the example in question. This can be done relatively easily in the middle of the enumeration by shifting only the tree and the word table.

I realised quite early on that there was not much point in using column 1 of the word table to store the coincidence words, as is done in SUBGROUP. This just confuses matters as location (COL+1, ROW) in WRDTBL corresponds to location (COL, ROW) in SPACE. Therefore I decided to store the coincidence words in the last column of the word table. I also changed the symbol used to represent the empty word from 1 to $\emptyset$. This makes it easier to take the inverse of the empty word, since $-\emptyset=\emptyset$. In SUBGROUP each -1 had to be changed to 1 before being written into the word table. Another advantage of this is that we can start numbering the subgroup generators from 1, instead of 2, which is less confusing if you wish to match up the actual subgroup generators with their numbers in the final presentation.

Initially I thought about introducing more values for CNTRL to determine the type of modified algorithm to be carried out. However, I decided that the easiest solution was to use the same values of CNTRL as for the ordinary Todd-Coxeter but to introduce two logical variables, MTC and EXTRA, which specify whether or not we are running the modified algorithm, and whether or not we are collecting the "extra relators". Thus, when we call MENMFL and MENMTL, we need only open FOR030.DAT and FOR031.DAT if EXTRA=TRUE.

I introduced a new facility to MODTC as an experiment to try to get the coset table to close more quickly. I took a complete or partial coset table and, using the

same methods as those employed in COSDEF (see page 89), worked out a set of possible coset definitions. I then "rubbed out" all of the other entries in the coset table except these original definitions. If FELSCH=TRUE, these definitions are also placed in the deduction queue and if MTC=TRUE, the first NALIVE rows of the word table are initialized with zeros. The enumeration can then be restarted from this point with NALIVE cosets already defined.

To start a coset enumeration in this way, I added the command OD to those in OPT. This finds the coset definitions then calls the appropriate enumeration routine – ENUMFL, ENUMTL, MENMFL or MENMTL – depending on the values of FELSCH and MTC. This enumeration routine is called with RENTER=1. Originally RENTER=$\emptyset$ meant an enumeration was to be restarted from scratch and a non-zero value of RENTER meant an enumeration was to be resumed from a partial coset table. Now I have changed it so that :

RENTER $< \emptyset$ starts from a coset table of definitions,
RENTER $= \emptyset$ starts from scratch
and RENTER $> \emptyset$ resumes an enumeration from a partial coset table.

I had to distinguish between resuming an enumeration from a partial coset table and from a set of coset definitions, as a number of variables have to be initialized in the second case, since no subgroup generator or relator processing has been done. Also, files FOR030.DAT and FOR031.DAT have to be opened if EXTRA=TRUE.

If a modified algorithm enumeration does not complete, and, for example, we add some more relations and start the enumeration again, the files FOR030.DAT and FOR031.DAT will be already open. Thus, before we open these files for enumerations where RENTER $\leq \emptyset$ and EXTRA=TRUE, we check to see if they are open already, and if they are, we close them.

I have made another alteration to the program in the case where an enumeration is resumed from a partial coset table. TC2 assumed in this case that all of the subgroup generators had been processed, and started on the relator defining phase. This may not always be the case, so now we check if SUBGRP=TRUE and start on the subgroup generation phase if it is. When we resume an enumeration KN is always started at 1 and FREELM at $\emptyset$ so, although we may waste some time covering the same ground twice, no errors can be made. If we run MENMTL or MENMFL and find that there are no subgroup generators we obviously cannot carry out the modified

algorithm. If this happens, a message to that effect is printed, and the routine exited.

Now that I have described the global changes to TC2 I will outline the important alterations to each main subroutine.

MENMTL

The subgroup generators are processed using MAPPLY, the modified version of APPLY. We then reintroduce the variable OLDNXT from SUBGROUP which saves the last "useful" value of NEXT. There are various places where the parts of the tree from OLDNXT onwards can be overwritten - I will discuss these later when I come to them.

The code to apply the HLT modified algorithm to the group relators is basically the same as that in ENUM of NEWREL. The forward scan is carried out first, and, if this does not close, we start a backward scan. However, this time, LOW and HIGH are set to $\emptyset$ initially, instead of 1.

If a forward scan completes we can set :

JOIN=LOW

and not (-HIGH, LOW) as explained on page 109. This is valid because HIGH always equals $\emptyset$ since MENMTL does not process the subgroup generators.

If a backward scan completes with a deduction, and the inverse of the deduction cannot be filled into the coset table because there is already a value there, we set :

OLDNXT=NEXT .

Thus we now cannot lose the tree entries between the original value of OLDNXT and the present value of NEXT. These are needed to decode the word associated with the deduction already written into the coset table and, if our new coincidence turned out to be trivial, they would be overwritten. If we can fill the deduction into the coset table, we can now make WRDTBL(J, IFRONT)=-JOIN, whether or not JOIN is the empty word. This cuts out three lines of code from NEWREL.

If the backward scan does not close in the lookahead phase we set

NEXT=OLDNXT .

This can save a lot of space in the tree, especially if we do a lot of lookaheads which do not end in a deduction or a coincidence.

If a forward scan or backward scan completes without a deduction we have either a trivial or a non-trivial coincidence.

If it is trivial, i.e. IFRONT=IBACK, we have a relation for FOR030.DAT. This is written to FOR030.DAT by a subroutine WRIT30 if JOIN≠∅ and EXTRA=TRUE. WRIT30 contains almost the same code, line for line, as the part of SUBGROUP which writes subgroup relations to PRESN.DAT. Thus the relation is traced out from scratch, so we do not need the word JOIN concatenated from the word table, and can set NEXT=OLDNXT and overwrite part of the tree. If EXTRA≠TRUE we also set NEXT=OLDNXT since we will never write our concatenated word, JOIN, to the word table or to the files FOR030.DAT or FOR031.DAT .

If the coincidence is non-trivial, i.e. IFRONT≠IBACK, we call MCOINC, the modified version of COINC.

I have taken out all lines of code in which we check for INDEX1 to be true, and only if the enumeration completes with NALIVE=1 do I set INDEX1=TRUE. There is no benefit to be gained in checking for INDEX1 earlier. In the ordinary Todd-Coxeter, part of the coincidence routine can be skipped if we find that we have an index of 1. However, this would leave an inconsistent word table in the modified algorithm, so no shortcuts can be taken.

When the enumeration completes we call COMPAC to get rid of redundant rows and renumber cosets in sequence. (Obviously, COMPAC had to be changed so that it compacts the word table as well as the coset table.) Then we call the subroutine WRPRSN to write out the subgroup presentation to PRESN.DAT. This is almost the same as the code used to write relations to PRESN.DAT in SUBGROUP. We then open a file called WRDTBL.DAT, and to this we write the coset table and the word table. This is not really needed but is useful for checking purposes. If EXTRA=TRUE we then write

1 997

1 999

to the last two lines of FOR030.DAT and FOR031.DAT to show that they are complete. All of the statistics of the enumeration are output during this phase - the index, the values of MAXCOS, TOTCOS, the number of relations written to PRESN.DAT, FOR030.DAT, FOR031.DAT and the size of the tree. Then all these files are closed.

Notes

If we process some of the subgroup generators more than once, i.e. in both the defining and the lookahead phase, and they are closed in both, then we will get more extra relations in FOR030.DAT if EXTRA=TRUE . This does not matter.

MAPPLY

This subroutine is called with the number of the subgroup generator SGEN, and HIGH is set to SGEN initially. If we are not processing subgroup generators, the value $\emptyset$ can be passed for SGEN. The statements to carry out the enumeration and concatenate the h-words are the same as those in MENMTL except that, if a forward scan completes, we must set :

JOIN=CATCON(-HIGH, LOW)

since HIGH can be non-zero.

MENMFL

This subroutine took a lot more time to write, since it was necessary to work out where and how the word table concatenations should be done - it was not quite as straightforward as I originally thought it would be.

First of all I cut out all the code which enables you to treat relators as subgroup generators. I decided that since these are redundant subgroup generators anyway, they would just introduce redundant entries into the tree, so there was no point in leaving in the option.

I added in the lines SUBGRP=TRUE at the beginning and SUBGRP=FALSE at the end of the subgroup generation phase, as these did not already exist. This means that if the subroutine is re-entered, we know whether or not we have completed the subgroup generation phase. Again MAPPLY is used to process the subgroup generators.

When a deduction is applied to a relator at the generator J, we start at J and work backwards, then forwards, trying to complete the relator cycle.

Here we start with the following variable values :

LOW=$\emptyset$

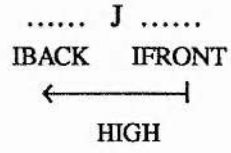
OLDNXT=NEXT

HIGH=WRDTBL(JI, IFRONT)

KT=IFRONT

where JI is the column number belonging to the inverse of the generator in column J, and KT is a variable used to mark the coset reached at either end of the relator.

Thus we have :



If the relator is of length 1 we immediately concatenate (-HIGH, LOW).

If IFRONT=IBACK, and EXTRA=TRUE, we call WRIT30 with coset number KT. This applies KT to the relator and writes the extended subgroup relator to FOR030.DAT. NEXT is then set to OLDNXT, whether EXTRA=TRUE or not.

If IFRONT≠IBACK we have a coincidence :

JOIN . IFRONT = IBACK

i.e. -WRDTBL(JI, IFRONT). IFRONT = IBACK, in this case, so we call MCOINC.

If the relator is not of length 1 we continue the backscan as far as we can go and concatenate HIGH with WRDTBL(JI, IBACK) each time. If we reach the beginning of the word during the scan we set KT=IBACK and continue our backward scan from the end of the relator with coset IBACK. If the relator closes on the backward scan we call CATCON(-HIGH, LOW). The result is written to the variable JOIN.

If IFRONT=IBACK and EXTRA=TRUE we call WRIT30 with coset KT, and NEXT becomes OLDNXT, whether EXTRA=TRUE or not.

If the relator closes with IFRONT≠IBACK, we have a non-trivial coincidence :

JOIN . IFRONT = IBACK

so we call MCOINC.

If the relator does not close on the backscan we commence the forward scan from the original deduction, concatenating LOW with successive values of WRDTBL(J, IFRONT), and setting LOW=JOIN each time. If we reach the end of the relator in this scan we set KT=IFRONT and carry on scanning from the beginning of the relator with IFRONT.

If this scan closes completely we call CATCON(-HIGH, LOW) and process the coincidence as before.

If the scan has a gap of length 1, i.e. we have a deduction, we call CATCON(-HIGH, LOW). Then we can fill the resulting word, JOIN, into the word table:

WRDTBL(J, IFRONT)=-JOIN, and its inverse,

WRDTBL(JI, IBACK)=JOIN.

[It is impossible for any information to be found on the forward scan which helps the backward scan to proceed, since we do no defining during the scanning. Thus, we can fill in the inverse right away, unlike the subroutine HLT.]

If the scan does not close, we set NEXT=OLDNXT and go on to the next "essentially different position".

When the enumeration finally completes, we follow exactly the same procedures as in MENMTL.

MCOINC

Unfortunately in TC2, the variables HIGH and LOW are used in subroutines COINC and PCL12 to denote the pair of coincident cosets. Thus, in these two routines, I used JHIGH where the variable HIGH was used in SUBGROUP.

The word concatenation parts of the routine are very similar to those in COINC in SUBGROUP, although, obviously, the first two columns of the coincidences are processed in MPCL12 instead. The only other alteration is that, after I make SPACE(I, LOW)=LOW, if LOWI=HIGH, I also set WRDTBL(I, LOW)=ICOINWR. This ensures that each time a coset table entry is inserted, a corresponding entry is filled into the word table and fixes the bug discussed in Section 2.4.5 (1).

MPCL12

This routine required the addition of many statements. I also introduced two new variables, ICWR1 and ICWR2, to store the coincidence words connected with the two coincidences stored in the variables LOW1S, HIGH1S and LOW2S, HIGH2S respectively. These, therefore, also have to be incorporated into the subroutine MNTCOIC, the modified version of NTCOIC .

The trickiest part of writing MPCL12 was encountered when dealing with the section which fills the current ultimate values for LOW and HIGH into the first two columns of their respective coincidence chains. This piece of code causes complications, because a different coincidence word has to be filled into the last column of the word table for each of these cosets.

The loop used to fill in the current irredundant values was changed to the following code - the additional statements are shown in italics :

```
40      LOWSH = -SPACE(1, LOWS)
        ICWR = ∅
        LOWC=LOWS
45      CALL CATCON(WRDTBL(NCOL+1, LOWC), ICWR, TREE)
        ICWR = JOIN
        LOWC = -SPACE(1, LOWC)
        IF (LOWC.NE.LOW) GO TO 45
        SPACE(1, LOWS)=-LOW
        WRDTBL(NCOL+1, LOWS) = ICWR
        IF ( LOW-LOWSH ) 50,60,60
50      LOWS = LOWSH
        GO TO 40
```

Thus, we end up with ICWR.LOWS=LOW for each successive value of LOWS and this value ICWR can be filled into WRDTBL(NCOL+1, LOWS).

Exactly the same procedure is followed for the HIGH chain.

In the process of finding the ultimate values of HIGH and LOW we concatenate the coincidence words down each chain and end up with

JHIGH.HIGH=LOW .

This part is the same as that in COINC in SUBGROUP.

If our ultimate values of HIGH and LOW turn out to be equal, we check that JHIGH≠∅ and if EXTRA=TRUE we write the high-numbered generator, JHIGH, to the file FOR031.DAT. I examined the possibility of expanding this relator before writing it to the file but decided that it was impossible to keep track of how it was built up.

If HIGH and LOW are not equal we mark HIGH as coincident to LOW. This is done by :

SPACE(1, HIGH)=-LOW

and WRDTBL(NCOL+1, HIGH)=JHIGH.

If LOW > HIGH we swap the values of HIGH and LOW and set JHIGH=-JHIGH.

Now ICOINWR=JHIGH and we look at columns 1 and 2 of SPACE for HIGH≡LOW. The code for this is the same as that used to process columns 3 to NCOL in MNCOINC.

Then we remove HIGH from the stored up cosets to be processed. We check first if LOW1S=HIGH. If it is, we replace ICWR1 by CATCON(ICOINWR, ICWR1), and replace LOW1S by LOW. Similarly, if HIGH1S=HIGH, ICWR1 is replaced by CATCON(ICWR1, -ICOINWR). We now check LOW1S≠HIGH1S. If LOW1S=HIGH1S we write ICWR1 to FOR030.DAT, if EXTRA=TRUE. LOW2S and HIGH2S are then treated in a similar fashion.

MNTCOIC

This is the modified algorithm version of NTCOIC which stores the new coincidences which are found during the processing of the first two columns in the coset table of other coincidences.

If we find that the coincidence which has just been found has already been noted, we obtain another relation for FOR031.DAT. ICWR is the coincidence word passed to the routine, i.e. ICWR.HIGH=LOW, so now, if HIGH=HIGH1S and LOW=LOW1S, we get the relation ICWR1=ICWR. Thus, if EXTRA=TRUE, we concatenate ICWR1 and -ICWR and write the result to the file FOR031.DAT.

Similarly, if HIGH=HIGH2S and LOW=LOW2S, we call CATCON(ICWR2, -ICWR) and write the result to FOR031.DAT.

There is another place in this routine where an extra relator could possibly be found. This occurs when LOW=HIGH on calling the routine and in this case ICWR itself is a relator and is written to FOR031.DAT.

WRPRSN

This is a modified algorithm routine to rewrite the subgroup generators and relators using the word table and write the final relations to PRESN.DAT. This

routine also writes the tree to TREE.DAT.

The subgroup relators are traced out in much the same way as those in SUBGROUP, there of course being different variable names and more complicated loops, due to the fact that the group relators can be stored with exponents > 1.

The tree is written to TREE.DAT, 10 integers to a row, each entry taking up 6 characters. The first line of TREE.DAT, however, is different. It has been altered to hold the values of the variables IDENT, NEXT-1 and GENEND. The first of these variables helps us to match up the tree with the appropriate file PRESN.DAT, assuming that an identifier has been given to the example. The second variable tells us the size of the tree, and the third indicates where the high-numbered generators start. I decided to do this instead of writing GENEND to both FOR030.DAT and FOR031.DAT.

WRIT30

This writes out the extra relators to FOR030.DAT. It is based on WRPRSN. First of all it checks to see if we are tracing out a subgroup generator, and if we are, -SGEN is written as the first element of the relator. From this point onwards the subgroup generator is treated in exactly the same way as a relator.

CATCON

This has been copied, almost line for line, from SUBGROUP. It has been slightly simplified now that the empty word is represented by $\emptyset$ instead of 1.

Some of the other subroutines have also been altered to give more flexibility with the modified algorithm.

ADDREL

I changed this so that I could add ARG new relators at a time instead of just one. If one of the new relators added is an involution, two columns will be used for it in the coset table. If ARG < $\emptyset$ the enumeration is not continued, otherwise the HLT method is applied to all of the new relators for each active coset. [Originally, the

subroutine always traced out the relations read in for every active coset.]

ADDSUB

Here we can add ARG new subgroup generators and trace out each new subgroup generator with coset 1.

If the result is coset 1 we do not add this generator to the rest of the subgroup generators if we are carrying out the ordinary Todd-Coxeter; however, if we are doing the modified algorithm we always add it. In this case we must alter the size of the tree, since PTEND will have changed, and if we have overwritten part of the tree, the enumeration must be started from scratch. If the new number of subgroup generators, NSGPG, is still less than GENEND we do not have to alter the existing structure of the tree, otherwise we must shift the tree up two places increasing GENEND and NEXT by 2. Then we look at each entry in the tree in turn. If its absolute value is greater than the old value of GENEND we increase it by 2. We then treat each entry in the word table similarly. Obviously, all of the relations in FOR030.DAT and FOR031.DAT are now inconsistent with the tree. I decided to discard these by opening new files for the extra relations.

If the subgroup generator traces out to a coset other than 1, the coincidence procedure is called if MTC=FALSE. Otherwise we call MAPPLY which traces out the subgroup generator again and calls MCOINC.

If the subgroup generator traces out to an unfilled position in the coset table we call APPLY or MAPPLY. If we still cannot define right round the subgroup generator, we set variable RENUM=TRUE and SUBGRP=TRUE, then, when the reading in of the other subgroup generators has finished, the enumeration is restarted automatically from the subgroup generation part of the defining phase. In TC2, if we could not define right round a subgroup generator, it was discarded.

DELREL

This was changed so as to disallow the deletion of an involutory relator if its associated generator is only occupying one column of the coset table.

Increasing the size of the tree automatically

If the tree runs out of space in the enumeration routines it is an easy matter to move the tree and the word table (assuming there is space left in Y) and restart the enumeration. However, if the tree overflows in the coincidence routines this is not really possible. In this case, if the tree and word table are shifted, you have to return to the main program in order to identify the arrays WRDTBL and TREE with their new locations in Y, then it is very difficult to continue from the point you left the coincidence routine.

In order to know whether or not the enumeration can be resumed in the case of the tree overflowing, I introduced alternative return points to the enumeration and coincidence subroutines. One of these is used for overflows where the enumeration can be resumed, the other for when we have to restart it from scratch. These return points are referenced by statement labels, and these appear in the list of parameters in the subroutine calls, prefixed by an asterisk.

E.g. in subroutine OPT we have :

```
CALL MENMTL(1,MAX,NCOL,SPACE,WRDTBL,TREE,*154,*151).
```

Thus, when executing MENMTL, if we come across a statement RETURN 1, we return to the statement labelled 154 in OPT, if RETURN 2, we go to statement 151.

In the enumeration routines I always ensure that I call CATCON before filling entries into the coset table. Thus, if I find that the tree is full, and cannot fill anything into the word table, I know that the coset table will still be consistent with the word table.

If the enumeration is being carried out from the main program instead of OPT, and the tree overflows, we call OPT and carry out the enumeration again from scratch. This time, when the tree runs out of space, it will be increased in size automatically.

Options added

I have added four new options to those in OPT, namely MT, AT, OD and PW. These have not simply been added as options 34 to 37, since the order of the options depends on whether they require compaction to be carried out on the coset table first, or whether they need a minimal spanning tree constructed.

MT This option changes to/from the modified algorithm, i.e. alters the values of MTC and EXTRA. These depend on the input parameter, PARMTR.

$$\text{PARMTR} = \begin{cases} -N & \text{sets MTC to TRUE and EXTRA to TRUE} \\ \emptyset & \text{sets MTC to FALSE} \\ N & \text{sets MTC to TRUE and EXTRA to FALSE} \end{cases}$$

If MTC is changed from FALSE to TRUE, we must calculate WRDBEG and TREBEG, then we must return to the main program so that we can call OPT again, identifying Y(WRDBEG) with the start of the array WRDTBL and Y(TREBEG) with the start of the array TREE.

AT This alters the size of the tree for the modified algorithm. If the parameter N is positive, it changes TREESZ to the smallest odd integer ≥ N. We then calculate TREBEG and NMAX. Using NMAX we find the location of WRDBEG and initialize NEXT to be GENEND and WRDEND to be WRDBEG.

If the parameter is negative, TREESZ is increased as a proportion of the available space in Y, depending on the current sizes of both the coset table and the tree. The tree is increased by DIFF (or DIFF+1 if DIFF is odd) where :

DIFF = REAL(TREESZ)/REAL(TREESZ+NALIVE*NCOL)*(WRDBEG-LASTPS-1) .

This is also what happens if tree overflow is detected during the enumeration routines. (If DIFF turns out to be < 10 we do not bother increasing TREESZ.) We then calculate the new values of TREBEG and NMAX and from these WRDBEG and WRDEND. Then the arrays WRDTBL and TREE are moved to their new positions, one entry at a time.

In both cases we now return to the main program. The variable RESTOR is given certain values so that when we call OPT with the new first locations of the arrays WRDTBL and TREE, the enumeration is automatically resumed, or started from scratch, whichever is appropriate to the example in question.

OD As explained on page 113, this chooses a set of coset definitions for the cosets 2 to NALIVE from a partial or complete coset table and restarts the enumeration using these definitions.

PW writes out the word table if MTC=TRUE. If PARMTR=∅, all of the word table is printed, if PARMTR=n, the first n rows are.

When we read in the initial data from a file or the terminal we must now read in the values of MTC and EXTRA. Then, when we save the coset table to a file, we now have to save the values of the additional variables MTC, EXTRA, GENEND, NEXT, WRDBEG, WRDEND, TREBEG and TREESZ as well. If MTC is TRUE we also want to write out the word table and the tree. Similarly, in the main program, where we read in the data from such a file, we must read in all of this information. If the enumeration is not a modified Todd-Coxeter, these variables will exist with initial value FALSE if they are logical, 1 if they are integer variables. These are set in BLOCK DATA and continue until over-ridden, either by the user via the options in OPT, or automatically by the program when carrying out the enumeration.

§3.3 An Adaption of the Program TTRANS - NTTRANS

TTRANS used the relative access file TREE.DAT to read in selected entries for decoding the high-numbered generators resulting from SUBGROUP. With SUBGROUP it is possible that only a few of the tree entries are required. However, to decode the relations in FOR030.DAT and FOR031.DAT, the tree is referenced more often. Thus, I decided it was easier to make the tree an integral part of GEN and read it into GEN before the relators. It is for this reason that MODTC writes the tree to a sequential file, instead of a relative access file. The size of the tree, TREESZ, is given a default value of 10001 and the total size of the array, TOTSIZ, is set to 1010001.

We read in the actual value of TREESZ from the first line of TREE.DAT, and calculate GENSIZ, the size of the array available to hold the relators and associated information, to be TOTSIZ-TREESZ. The tree is then written into GEN starting at GEN(GENSIZ+1). TREE.DAT is then closed, so that it can be used by other programs. This is a problem with TTRANS on the VAX, since the relative access file TREE.DAT has to be left open throughout the execution of the program. [Note that there always has to be a TREE.DAT available to read in, whether or not it is necessary to the presentation in question.] The new file TREE.DAT also contains the values of

IDENT and BIGGEN. This makes it easier to match up the files TREE.DAT with their corresponding PRESN.DAT files.

Since the format of the tree has been changed, all statements referencing the tree have had to be altered right through the program. I also altered the NF command which did not work properly in all cases. Originally, if the file was not PRESN.DAT, the first line of the file was read in as a relator. As long as the field for the identifier was empty this did not produce an error. However, one fewer actual relator was read in. I changed this so that the information from the first line was read in the first time that the file was accessed. I also decided to use the command NF with no argument to read in all of the relators from the current input file. Before, this operation could only be achieved by inputting a very large positive argument and hoping that there were fewer relators than that in the file ! In TTRANS the permissible range of numbers for data files was 10 to 30 inclusive (excluding 12). I had to increase the upper limit to allow for the extra relation files. In fact I increased it slightly further to 34.

I added eight new options to NTTRANS – WC, AE, AD, AN, TR, SW, WT, and SQ .

WC is the weighted substring search version of DC.

AE carries out the same process as EH but executes the equal length substitutions and substring searches IV, SF, IV-9, SF every NTIMES eliminations. The variable NTIMES is initially set by default to 50 but can be altered by the user. The substitution sequence is repeated until there is no change in the number of relators and their combined length. This procedure helps to prevent the presentation becoming very long if a large number of generators have to be decoded from the tree.

AD is equivalent to DC but with IV, SF, IV-9, SF sequences executed every NTIMES eliminations.

AN alters NTIMES to IABS(ARG).

WT alters the weights on the generators. If ARG≤∅ the user is asked for a generator to be changed, and its new weight. The loop is exited by giving ∅ as the next generator number to be altered. If ARG>∅ the weights are altered as follows :

ARG=1 all weights are set to 1.

ARG=2 the weight of generator I is set to I.

ARG=3 the weight of generator I is set to 100+I*2.

ARG=4 the weight of generator I is set to 10.

ARG=11 the weight of generator I is set to 1, I<BIGGEN.

ARG=12 the weight of generator I is set to I, I<BIGGEN.

The other numbers 5-10 and 13-20 inclusive have been left unassigned for the user to add more options.

SQ This defines sequences of substring searching and equal length substitutions.

SQ1 IV, SF, IV-9, SF.

SQ2 IV2, SF, IV, SF, IV-2 , SF, IV-9, SF.

SQ3 SQ1 followed by SQ2.

SQ4 SQ3 repeated until there is no change in the number of the relators and the overall length of the presentation.

I also added two new facilities for decoding generators and substring searching in word mode.

TR Copies words written in high numbered generators in unit ARG into RELLST, and decodes them using the tree, down to generators less than BIGGEN.

SW Here relations in file ARG are read into the list of elimination rules and used to try and shorten the words in RELLST with substring substitutions. The user is asked for a file number to which to write the output.

Examples of results which I have obtained with the last two programs, MODTC and NTTRANS, are given in the next chapter.

Chapter 4

Results Obtained from the Computer Programs

As mentioned before in Section 2.1, statistics for a coset enumeration can be altered considerably by seemingly trivial changes such as rearranging the order of the relators, or cyclically permuting individual relators. Likewise, the complexity of the high-numbered generators in the final presentation obtained from the modified Todd-Coxeter algorithm can be affected quite drastically by the type of enumeration done, the number of redundant cosets defined etc.. The decoding of this presentation back into the original subgroup generators using NTTRANS can also be considerably affected by the frequency of substring searching and the alteration of the exact points at which this substring searching is carried out. Surprisingly, substring searching after each elimination, apart from being very time consuming if the presentation is lengthy, does not always give the best results.

The structure of the relators in the presentation itself also affects the success of the simplification, due to the way that substring searching is carried out. Since relators are stored as subwords with an exponent, and only this subword is searched for matching substrings, many potential substitutions can be missed. An example given later highlights this point.

The data in this chapter was all collected from my programs running on a SUN 3/260. However, I will continue to use the VAX FORTRAN names for files such as PRESN.DAT, TREE.DAT and FOR030.DAT as I used in Chapters 2 and 3. All execution times are given in seconds to two decimal places. Thus, a time of 0.00 indicates that the command took less than 0.01 seconds to execute. For most examples two different times are given. The first is the elapsed user time, i.e. the amount of time actually used by the computer to carry out the calculation, and the other is the elapsed system time which is the time taken to do other tasks, such as writing to the screen. If the same calculation is run more than once, the times obtained will vary slightly from run to run. This variation can be as much as 10 seconds for runs which take a few thousand seconds, therefore there is no point in finding the times more accurately. The main reason for timing certain runs is to compare roughly the relative speeds of different methods of enumeration, decoding and simplification for the same example.

Note that, in the output from MODTC, the number of relators written to PRESN.DAT, FOR030.DAT and FOR031.DAT is the total number of non-trivial relators found, and not the number of distinct relators. In some cases, these three sets of relators may be highly redundant.

Firstly I would like to show how the frequency of substring searching can affect the example. I have chosen one in which no relators have exponents greater than 1, thus eliminating any effect due to the way NTTRANS substring searches exponentiated relators. The example I have chosen is that used as a test example in the paper on the modified algorithm [2].

Example 1

$$G = \langle a, b \mid a b^2 a^{-1} b^{-1} a^3 b^{-1} = b a^2 b^{-1} a^{-1} b^3 a^{-1} = 1 \rangle$$

$$H = \langle [a^{-1}, b^{-1}], [a^{-1}, b], [a, b] \rangle$$

Using MODTC, with no restriction on the maximum number of cosets which can be defined, we obtain the following :

Index = 18 Max = 45 Total = 49
39 relations written to file PRESN.DAT
The maximum tree size is 110

The presentation in PRESN.DAT was read into NTTRANS and the resulting 35 relations decoded, using in turn the commands EH1, EH2,..., EH15. [Note that 4 of the original 39 relations were duplicated, so disappeared.] In each case this command was followed by SF if a substring search did not automatically follow the last elimination. The number of relators present at this point and their combined length was noted. The command SQ1 was then applied repeatedly until there was no change in the size of the presentation. The results obtained were as follows, the total number of relators in each presentation and their combined length given as an ordered pair (no, length) :

No. eliminations between substring searches (n)	Presentation after EH(n) [and SF if needed]	Presentation after SQ1
1	(18, 208)	(10, 64)
2	(14, 140)	(12, 82)
3	(20, 270)	(10, 64)
4	(20, 274)	(13, 80)
5	(18, 244)	(12, 76)
6	(20, 266)	(17, 178)
7	(20, 284)	(10, 64)
8	(20, 308)	(13, 78)
9	(19, 274)	(19, 236)
10	(20, 286)	(20, 206)
11	(20, 316)	(12, 88)
12	(20, 280)	(19, 202)
13	(20, 324)	(11, 66)
14	(20, 324)	(11, 66)
15	(20, 278)	(18, 230)

At this point it is worth noting that the number of eliminations carried out between successive substring searches is not always 'n', since the variable which keeps count of how many eliminations have been carried out is incremented after each elimination, by an amount which depends on the length of the relator used for the elimination.

For comparison I tried the command EH1000 followed by SF, which ensured that no substring searching was done between eliminations, (only 16 eliminations are done in the decoding). This resulted in a presentation with 20 relators and combined length 314 which, after some applications of SQ1, was reduced further to length 280, there being no change in the number of relators.

From the above table it can be seen that there is no correlation between the size of the final presentation obtained and the value of n. The best value of n to use varies from example to example and can only be found by trial and error. I usually use n=10. In this case the substring searching is frequent enough to stop the presentation from growing too large between substring searches, but not carried out often enough to slow the decoding down too much. Obviously with a small presentation like this, it

is not too difficult to eliminate redundant relators from the final presentation, especially if the corresponding group is finite; however, if the final presentation has a combined length of several thousand, this is not such an easy task. In cases like this it is often impractical to try the decoding with many different values of n and select the best, since it may take hours for the computer to decode and simplify each presentation. This is where the "extra relations" described in Section 3.1 come into their own.

For instance, in the above example, if we run MODTC again, collecting extra relations, we obtain 19 relations in FOR030.DAT and 1 in FOR031.DAT. These files were decoded separately with DC-10 followed by SF, then simplified with SQ1 until there was no change. The resulting relations from file FOR030.DAT were then written to the checkpoint file FOR017.DAT and the relation in FOR031.DAT to FOR018.DAT.

At this point we have 19 relations with total length 260 in FOR017.DAT and 1 relation with length 108 in FOR018.DAT. These were then combined with the relations obtained from PRESN.DAT using n=10, substring searched and simplified with SQ1 until there was no change. The resulting presentation has 15 relators and total length 112 - an improvement on (20, 206). If the first 10 relations only in FOR017.DAT are added to those obtained from PRESN.DAT, a final presentation with 11 relators and total length 72 is obtained. This is very comparable to the best presentation obtained in the above table.

Example 2

In [4] the subgroup $\langle x, a^8 \rangle$ in the group $\langle x, a \mid x^{-1}a^2x = a^3 \rangle$ was considered. A total of 8 cosets have to be defined before complete collapse is achieved. The presentation obtained for the group on the new generators was :

$$\langle \alpha, \beta \mid [\beta,\alpha]^2 = \beta,\ [[\beta,\alpha],\alpha] = [\alpha,\beta]^\alpha\beta = \beta^{-1}[\beta,\alpha]^\alpha[\beta,\alpha] = (\beta^{-1}[\beta,\alpha]^{\alpha^2})^2 \rangle. \quad \ldots\ldots\ldots\ldots\ldots\ldots (1)$$

where $\alpha = x$ and $\beta = a^8$ as defined in [4]. Substituting $[\beta,\alpha]^2$ for β (from the first relation) into $[[\beta,\alpha],\alpha] = [\alpha,\beta]^\alpha\beta$ we obtain $[\beta,\alpha] = [[\beta,\alpha],\alpha]^2$. Then,

$$\begin{aligned} & \beta^{-1}[\beta,\alpha]^\alpha[\beta,\alpha] \\ = \ & [\beta,\alpha]^{-2}\alpha^{-1}[\beta,\alpha]\alpha[\beta,\alpha] \qquad \text{since } \beta = [\beta,\alpha]^2 \\ = \ & [\beta,\alpha]^{-1}[[\beta,\alpha],\alpha]\,[\beta,\alpha] \end{aligned}$$

$= [[\beta,\alpha],\alpha]^{-2}[[\beta,\alpha],\alpha]\ [[\beta,\alpha],\alpha]^2$ since $[\beta,\alpha]=[[\beta,\alpha],\alpha]^2$

$= [[\beta,\alpha],\alpha].$

Thus, the relation $[[\beta,\alpha],\alpha] = \beta^{-1}[\beta,\alpha]^{\alpha}[\beta,\alpha]$ is redundant, so we have the shorter presentation :

$$\langle\ \alpha, \beta \mid [\beta,\alpha]^2=\beta,\ [[\beta,\alpha],\alpha] = [\alpha,\beta]^{\alpha}\beta = (\beta^{-1}[\beta,\alpha]^{\alpha^2})^2\ \rangle. \qquad (2)$$

By machine, carrying out the enumeration using MODTC, we obviously get only three relators written to the file PRESN.DAT, two of these resulting from the subgroup generators, the other from the group relation. The first subgroup generator, x, gives us the information $\bar{1}x = \alpha.\bar{1}$. This will always occur in the final coset table since coset 1 cannot disappear due to coincidences. Thus, unless the enumeration is carried out by processing the relator before the first subgroup generator, (which is very unlikely), the final relation obtained from this subgroup generator is trivial. Hence, from the modified algorithm we get at most two non-trivial relations.

The modified algorithm was carried out with MODTC using the HLT method, collecting extra relations, and the following data was obtained :

Index = 1 Max = 19 Total = 20
3 relations written to file PRESN.DAT
The maximum tree size is 88
File FOR030.DAT contains 0 relations
File FOR031.DAT contains 7 relations

Reading PRESN.DAT into NTTRANS, and using the command EH10 followed by a final SF to decode the high-numbered generators, we end up with a two relator presentation as expected. These two relations have combined length 247 and are :

$\alpha\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha^{-1}\beta^{-1}\alpha\ \beta\alpha^{-1}\beta^{-1}\alpha^{-1}\beta$
$\alpha\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}$
$\alpha\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}\alpha^{-1}\beta$
$\alpha\beta^{-1}\alpha^{-1}\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha^{-1}$
$\beta^{-1} = 1$

and

$$\alpha^3\beta\alpha^{-2}\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}\alpha^{-1}$$
$$\beta\alpha\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha^{-2}\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha\beta\alpha^{-1}$$
$$\beta^{-1}\alpha\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha^{-2}\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha\beta^{-1}\alpha^{-1}$$
$$\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha\beta\alpha^{-1}\beta^{-1}\alpha\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha\beta^{-1}\alpha^{-1}\beta\alpha^{-1}$$
$$\beta^{-1}\alpha\beta = 1. \qquad \ldots\ldots\ldots\ldots\ldots\ldots \quad (3)$$

This presentation can not be simplified any further by any of the techniques described in Section 2.6 and, as it stands, is not terribly useful for most purposes, for example other coset enumerations. These two relations were written to a checkpoint file FOR014.DAT.

Then NTTRANS was restarted, and the seven relators in FOR031.DAT read in. These were decoded using the command DC-10 (again followed by SF) to obtain a final set of five relations with total length 63.

$$\left.\begin{aligned}
&\alpha\beta^3\alpha^{-1}\beta^{-2} = 1\\
&\alpha\beta\alpha^{-1}\beta\alpha\beta^{-1}\alpha^{-1}\beta^{-1} = 1\\
&\alpha^2\beta\alpha^{-2}\beta\alpha^2\beta^{-1}\alpha^{-2}\beta^{-1} = 1\\
&\alpha^3\beta\alpha^{-3}\beta\alpha^3\beta^{-1}\alpha^{-3}\beta^{-1} = 1\\
&\alpha^3\beta^{-1}\alpha^{-1}\beta\alpha^{-2}\beta^{-1}\alpha\beta\alpha^2\beta\alpha^{-1}\beta^{-1}\alpha^{-1}\beta^{-1}\alpha^{-1}\beta = 1
\end{aligned}\right\} \qquad (4)$$

Again these cannot be simplified further by machine and were written to the file FOR015.DAT.

On putting together files FOR014.DAT and FOR015.DAT, then substring searching, I obtained a presentation with 7 relators and total length 114. This can be further reduced using SQ1 repeatedly to obtain 6 relators with total length 83 :

$$\left.\begin{aligned}
&\alpha\beta^3\alpha^{-1}\beta^{-2} = 1\\
&\alpha\beta\alpha^{-1}\beta\alpha\beta^{-1}\alpha^{-1}\beta^{-1} = 1\\
&\alpha^2\beta\alpha^{-2}\beta\alpha^2\beta^{-1}\alpha^{-2}\beta^{-1} = 1\\
&\alpha^3\beta\alpha^{-3}\beta\alpha^3\beta^{-1}\alpha^{-3}\beta^{-1} = 1\\
&\alpha^3\beta\alpha^{-1}\beta\alpha^{-2}\beta^{-1}\alpha\beta^{-1}\alpha^2\beta^2\alpha^{-2}\beta^{-1}\alpha^{-1}\beta = 1\\
&\alpha^3\beta^{-1}\alpha^{-1}\beta\alpha^{-2}\beta^{-1}\alpha\beta\alpha\beta^{-1}\alpha\beta\alpha^{-2}\beta^{-1}\alpha^{-1}\beta = 1
\end{aligned}\right\} \qquad (5)$$

Alternatively, if we combine files PRESN.DAT and FOR031.DAT in the first instance, and decode using DC-10, SF and simplify via SQ1 until there is no further change, we obtain a presentation with four relators and total length 43. [This is in fact

the best that I could obtain by any method.]

$$\langle \alpha, \beta \mid \alpha\beta^3\alpha^{-1}\beta^{-2} = 1,\ \alpha\beta\alpha^{-1}\beta\alpha\beta^{-1}\alpha^{-1}\beta^{-1} = 1,\ \alpha^2\beta\alpha^{-2}\beta\alpha^2\beta^{-1}\alpha^{-2}\beta^{-1} = 1,$$
$$\alpha^3\beta\alpha^{-3}\beta\alpha^3\beta^{-1}\alpha^{-3}\beta^{-1} = 1 \rangle \qquad (6)$$

It can be readily seen that these relations are just the first four of those in presentation (5). These can be written more succinctly as :

$$\langle \alpha, \beta \mid [\beta^2,\alpha] = \beta,\ [\alpha,\beta][\alpha,\beta^{-1}] = [\alpha^2,\beta][\alpha^2,\beta^{-1}] = [\alpha^3,\beta][\alpha^3,\beta^{-1}] = 1 \rangle. \quad (7)$$

This presentation is infinitely more useful than (3) !

Splitting up the last set of relations in (2) into two relations with the shortest possible length we have :

$$\langle \alpha, \beta \mid [\beta,\alpha]^2 = \beta,\ [[\beta,\alpha],\alpha] = [\alpha,\beta]^\alpha\beta\ ,\ [\alpha,\beta]^\alpha\beta = (\beta^{-1}[\beta,\alpha]^{\alpha^2})^2 \rangle. \quad (8)$$

The total length of this presentation is 51, so the final presentation (7), obtained from MODTC and NTTRANS using the extra relations, is shorter than the one achieved by hand in [4].

I also attempted this example using the Felsch method. This time the statistics from the enumeration were :

Index = 1 Max = 8 Total = 8
3 relations written to file PRESN.DAT
The maximum tree size is 62
File FOR030.DAT contains 10 relations
File FOR031.DAT contains 6 relations

Using EH10, the relations in PRESN.DAT reduced to 2 relations with total length 293. Decoding file FOR030.DAT and FOR031.DAT using DC-10 yielded 1 relation with length 138 and 3 relators with combined length 49 respectively. These were added to those obtained from PRESN.DAT, but the resulting presentation could not be reduced further than 5 relators with total length 213.

However, if files FOR030.DAT, FOR031.DAT and PRESN.DAT were all reduced together with DC-10 and then SQ1 applied until there was no change, I obtained a presentation with 4 relators and total length 48 - not quite as good as (7) but still shorter than the hand calculations.

I tried the enumeration in many different ways with different values of MAXTAB, and decoded PRESN.DAT with different values of EH(n). The best

presentation I ever managed was 2 relators with total length 199. Thus, in this case, the extra relators have to be used to get a presentation with reasonable length.

In the rest of this chapter I will use the term unexponentiated length to denote the length of the subword, w, of a relator when the relator is written in the form w^n, $n \in N$. This is the definition of the term "length" used by TTRANS and NTTRANS. However, this definition makes it difficult to compare the relative sizes of two presentations for the same group, where one contains many relators with large exponents. In what follows I will give the size of each presentation as an ordered triplet if the unexponentiated length of the presentation is different from the total length. This will have the form :

(no. of relators, unexponentiated length, total length).

Example 3

The next example I would like to consider is also one in which the subgroup generators generate the group itself. This time I would like to look at

$$A_5 \cong \langle a, b \mid a^2 = b^3 = (ab)^5 = 1 \rangle$$

with the new generating set

$$\{x,y\} = \{ (ab)^4, (ba)^4 \}.$$

When a straight HLT enumeration with no MAXTAB is carried out in MODTC, collecting extra relations, we obtain the following :

Index = 1 Max = 14 Total = 14
5 relations written to file PRESN.DAT
The maximum tree size is 54
File FOR030.DAT contains 0 relations
File FOR031.DAT contains 4 relations

This time when PRESN.DAT is read in to NTTRANS and reduced using EH10 followed by SF, we get a presentation with 5 relators, total unexponentiated length 79 and total length 167.

$$\langle x, y \mid x^5 y x^{-5} y x^4 y^{-1} x^{-1} y x^{-4} y^{-1} = x^5 y^{-1} x^5 y^{-1} x^5 y^{-1} x^5 y^{-2} = (x^{13} y^{-1})^2 = (x^5 y^{-1})^5 = (x^9 y^{-1})^6 = 1 \rangle \quad (9)$$

This can not be reduced any further by machine using any of the commands in Section 2.6. However, it is obvious that the presentation can be simplified. For example, the second relator gives us $(x^5 y^{-1})^4 = y$ which can be substituted into the fourth relator to obtain $y x^5 y^{-1} = 1 \Rightarrow x^5 = 1$. This information can then be used to reduce the presentation (9) to :

$$\langle x, y \mid y x y = x y x, y^5 = (x^2 y)^2 = x^5 = (x y)^6 = 1 \rangle. \quad (10)$$

This apparently obvious substitution is not found by the computer because of the way in which the substring matching part of the substring searching procedure is carried out in NTTRANS. What happens in NTTRANS is that suitable matches are only searched for in the subword $x^5 y^{-1}$ of $(x^5 y^{-1})^5 = 1$ and not in the whole relation. In small examples such as this one, further simplification can be carried out by hand, but if this happened with two relators which were several hundred characters long, the fact that simplification was possible would not be so obvious.

Although in this example the fact that exponents are not collected gives a bad final answer, this is not the case in all examples. If exponents are not collected, the final presentation may not contain any "nice" relators, such as powers of generators. To get round this problem it may be a good idea to introduce a new variable to NTTRANS to control whether or not exponents are to be collected. This facility can then be turned on or off from within the program. If it is turned off the relators will all be expanded, so, in this example, the final stage of the simplification can be carried out with these expanded relators and the obvious substitutions will be found.

Decoding file FOR031.DAT with the command DC-10 followed by SF, we end up with relations (3, 10, 18). These are as follows :

$$x^5 = 1, \ (x y)^3 = 1, \ x y^2 x y^{-3} = 1.$$

When these relations are added to (9), a substring search carried out, and SQ1 applied until there is no further change, we obtain a presentation (4, 7, 26) :

$$\langle x, y \mid x^5 = 1, \ y^5 = 1, \ (x^2 y)^2 = 1, \ (x y)^3 = 1 \rangle. \quad (11)$$

Again this is very close to the best presentation that can be obtained on these generators.

I repeated this example using the Felsch method and obtained the following results from MODTC :

Index = 1 Max = 14 Total = 14
5 relations written to file PRESN.DAT
The maximum tree size is 52
File FOR030.DAT contains 30 relations
File FOR031.DAT contains 3 relations

When PRESN.DAT is read into NTTRANS, and decoded using EH10, we get a presentation with 5 relations, total unexponentiated length 24 and total length 76 :

$$\langle x, y \mid (x^2y)^2 = x^4y^{-1}xyx^{-4}y^{-1} = x^{15} = (x^2y^{-1})^6 = (x^2y^{-1}x^{-1}y^{-1})^5 = 1 \rangle.$$

This cannot be reduced any further.

Decoding file FOR030.DAT with DC-10 yields three relations with total unexponentiated length 29, total length 170 :

$$x^{20} = 1, \ (x^9y^{-1})^6 = 1, \ (x^9y^{-1}x^{-7}y^{-1})^5 = 1.$$

Then, decoding FOR031.DAT with DC-10, SF gives relations (3, 10, 18) :

$$x^5 = 1, \ (xy)^3 = 1, \ xy^2xy^{-3} = 1.$$

Combining these three sets of relations and simplifying with the commands SF, SQ4, RP yields a presentation (5, 10, 40). This is as follows :

$$\langle x, y \mid x^5 = y^5 = (x^2y)^2 = (xy)^3 = (x^2y^{-1})^6 = 1 \rangle.$$

I tried this enumeration and decoding with many different parameters, and only by using the extra relations could I consistently find the relation $y^5 = 1$. One or two combinations of parameters resulted in a presentation containing $y^5 = 1$ from PRESN.DAT alone, but no combination that I tried yielded <u>both</u> $x^5 = 1$ and $y^5 = 1$.

Thus, in this case, the extra relations not only help us find a short presentation, but give us additional relators which give us useful information about the group.

In the rest of this chapter I have chosen a mixture of test examples from papers in which a reasonably small presentation for the subgroup on the actual subgroup generators is given. Thus, I can easily check to see if my presentations are good or bad, and how closely they resemble the one given.

As a standard method of simplification to use after the decoding phase in NTTRANS I chose the command SQ4. [This will have the same effect as SQ1 used repeatedly if there are no involutory generators.] This sequence of commands does not include IV±3 or IV4. The former was omitted deliberately since equal length substitutions are performed at random, and therefore runs of an example cannot be repeated with the guarantee of obtaining identical presentations each time.

After SQ4 the command RP is usually used to check that there are not two or more powers of the same subword present as relators.

Example 4

$G = A_7 = \langle a, b \mid a^2 = b^4 = (ab)^7 = [a, b]^5 = (abab^2ab^{-1})^3 = 1 \rangle$

$H = \langle a^b, b^{ab^{-1}a} \rangle \cong A_6$

I have chosen this example from [16] to show how the file PRESN.DAT and the two sets of extra relations obtained from a modified Todd-Coxeter algorithm enumeration can be combined, and what sort of variation in results can be obtained. To achieve this I will carry out the enumeration in four different ways, two using the HLT method and two Felsch, then I will decode each of the sets of relations obtained with the commands EH10 (or DC-10), EH5 (or DC-5) and EH1 (or DC-1). These will each be treated separately and the extra relations added in different ways to the corresponding file obtained from PRESN.DAT. The shortest presentation obtained for each of the commands EH(n) is marked with an asterisk followed by the value of n. If two methods give the same best ordered triplet both will be marked.

This is a relatively small example which does not take very long to decode or simplify so was ideal to choose for this task. Obviously with larger examples even more variation in execution times and sizes of presentations will be observed.

Note that the input and result file numbers quoted in the following tables refer only to the files in the same table.

(i) AC-4, (no MAXTAB)

The results obtained from MODTC are :

Index = 7 Max = 474 Total = 569

29 relations written to file PRESN.DAT

The maximum tree size is 836

File FOR030.DAT contains 24 relations

File FOR031.DAT contains 73 relations

Elapsed user time = 0.48 Elapsed system time = 0.10

These files are then processed by NTTRANS as follows :

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time	Results File
PRESN.DAT	EH10	(8, 699, 1389)	1.22	0.12	14
14	SQ4	(7, 273, 577)	2.46	1.18	15
PRESN.DAT	EH5	(8, 699, 1389)	1.28	0.04	22
22	SQ4	(7, 273, 577)	2.40	0.22	23
PRESN.DAT	EH1	(8, 699, 1389)	1.76	0.20	26
26	SQ4	(7, 273, 577)	2.40	0.26	27
FOR030.DAT	DC-10	(2, 2, 6)	0.06	0.00	⊆15
FOR030.DAT	DC-5	(2, 2, 6)	0.02	0.02	⊆23
FOR030.DAT	DC-1	(2, 2, 6)	0.08	0.02	⊆27
FOR031.DAT	DC-10, SF	(28, 662, 666)	106.36	1.50	16
16	SQ4	(6, 34, 70)	1.32	0.20	17
FOR031.DAT	DC-5, SF	(23, 611, 647)	118.82	4.56	24
24	SQ4	(6, 18, 68)	3.14	0.34	25
FOR031.DAT	DC-1	(26, 562, 590)	230.28	5.92	28
28	SQ4	(9, 79, 168)	2.34	0.30	29
15 + 17	SF, SQ4, RP	(8, 50, 118)	1.34	0.34	
23 + 25	SF, SQ4	(7, 23, 118)	1.12	0.36	*5
27 + 29	SF, SQ4	(11, 87, 223)	1.78	0.24	

PRESN+30+31	DC-10	(33, 736, 897)	119.28	3.94	
	SQ4	(7, 20, 88)	3.20	0.26	* 10
PRESN+30+31	DC-5, SF	(25, 513, 623)	131.42	3.38	
	SQ4	(8, 30, 139)	1.76	0.30	
PRESN+30+31	DC-1	(26, 596, 708)	263.76	7.48	
	SQ4	(9, 76, 173)	3.30	0.36	
PRESN + 30	DC-10, SF	(10, 200, 370)	0.90	0.16	
	SQ4	(9, 173, 335)	1.26	0.30	
PRESN + 30	DC-5, SF	(10, 200, 370)	0.98	0.16	
	SQ4	(9, 173, 335)	1.22	0.26	
PRESN + 30	DC-1	(10, 209, 379)	1.40	0.32	
	SQ4	(9, 172, 334)	1.26	0.18	
PRESN + 31	DC-10, SF	(28, 736, 857)	108.40	1.86	
	SQ4, RP	(9, 77, 170)	4.06	0.22	
PRESN + 31	DC-5, SF	(29, 626, 755)	126.34	2.58	
	SQ4, RP	(9, 72, 169)	3.14	0.38	
PRESN + 31	DC-1	(19, 334, 410)	255.24	5.38	
	SQ4, RP	(6, 18, 68)	1.60	0.32	* 1
14 + 16	SF, SQ4	(8, 36, 158)	8.74	0.32	
22 + 24	SF, SQ4, RP	(8, 29, 117)	7.92	0.36	
26 + 28	SF, SQ4	(11, 87, 223)	6.56	0.92	

(ii) AC-4, AM250

Here I obtained the following from MODTC :

Index = 7 Max = 250 Total = 341

29 relations written to file PRESN.DAT

The maximum tree size is 478

File FOR030.DAT contains 39 relations

File FOR031.DAT contains 47 relations

Elapsed user time = 0.66 Elapsed system time = 0.10

Then from NTTRANS :

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time	Results File
PRESN.DAT	EH10	(8, 699, 1389)	0.92	0.16	14
14	SQ4	(7, 273, 577)	2.48	0.16	15
PRESN.DAT	EH5	(8, 699, 1389)	1.02	0.18	23
23	SQ4	(7, 273, 577)	2.62	0.10	24
PRESN.DAT	EH1	(8, 699, 1389)	1.52	0.12	19
19	SQ4	(7, 273, 577)	2.42	0.30	20
FOR030.DAT	DC-10	(2, 2, 6)	0.02	0.04	⊆15
FOR030.DAT	DC-5	(2, 2, 6)	0.02	0.00	⊆24
FOR030.DAT	DC-1	(2, 2, 6)	0.02	0.06	⊆20
FOR031.DAT	DC-10	(21, 522, 536)	23.12	0.86	16
16	SQ4	(6, 20, 65)	2.04	0.20	17
FOR031.DAT	DC-5	(19, 357, 371)	28.16	1.42	25
25	SQ4	(7, 66, 111)	1.34	0.28	26
FOR031.DAT	DC-1	(20, 451, 473)	52.56	2.92	21
21	SQ4	(7, 26, 92)	1.82	0.50	22
15 + 17	SF, SQ4, RP	(6, 18, 68)	0.94	0.30	* 10
24 + 26	SF, SQ4	(10, 81, 187)	1.78	0.30	
20 + 22	SF, SQ4	(9, 43, 159)	1.18	0.46	
PRESN+30+31	DC-10, SF	(22, 477, 599)	31.88	1.62	
	SQ4	(8, 70, 189)	2.60	0.24	
PRESN+30+31	DC-5, SF	(18, 332, 444)	35.16	1.52	
	SQ4	(9, 72, 169)	1.10	0.34	
PRESN+30+31	DC-1	(22, 448, 581)	65.12	1.98	
	SQ4, RP	(9, 104, 177)	2.30	0.30	
PRESN + 30	DC-10, SF	(9, 201, 371)	0.88	0.08	
	SQ4	(9, 170, 332)	1.00	0.28	
PRESN + 30	DC-5, SF	(9, 201, 371)	0.94	0.14	
	SQ4	(9, 170, 332)	1.10	0.46	
PRESN + 30	DC-1	(9, 203, 373)	1.56	0.18	
	SQ4	(9, 165, 327)	1.20	0.44	
PRESN + 31	DC-10, SF	(22, 426, 547)	30.78	1.06	
	SQ4	(9, 72, 209)	1.92	0.36	
PRESN + 31	DC-5, SF	(19, 486, 607)	33.74	2.10	
	SQ4	(7, 23, 93)	1.96	0.34	* 5
PRESN + 31	DC-1	(21, 365, 546)	60.00	2.16	
	SQ4	(9, 79, 168)	1.74	0.38	

14 + 16	SF, SQ4, RP	(7, 23, 93)	10.46	0.32	
23 + 25	SF, SQ4	(8, 62, 124)	7.30	0.50	
19 + 21	SF, SQ4	(7, 23, 93)	7.24	0.48	*1

(iii) AC-2, AS5

From MODTC I obtained :

Index = 7 Max = 275 Total = 280

29 relations written to file PRESN.DAT

The maximum tree size is 390

File FOR030.DAT contains 216 relations

File FOR031.DAT contains 44 relations

Elapsed user time = 2.72 Elapsed system time = 0.06

Then from NTTRANS :

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time	Results File
PRESN.DAT	EH10	(9, 814, 1639)	2.34	0.14	14
14	SQ4	(9, 392, 857)	5.70	0.46	15
PRESN.DAT	EH5	(9, 814, 1639)	2.38	0.16	20
20	SQ4	(9, 392, 857)	5.90	0.38	21
PRESN.DAT	EH1	(9, 814, 1639)	3.48	0.22	26
26	SQ4	(9, 392, 857)	6.22	0.38	27
FOR030.DAT	DC-10	(10, 359, 633)	2.82	0.20	16
16	SQ4	(10, 214, 518)	2.62	0.36	17
FOR030.DAT	DC-5	(10, 359, 633)	3.72	0.10	22
22	SQ4	(10, 214, 518)	2.58	0.40	23
FOR030.DAT	DC-1	(13, 426, 446)	26.24	1.76	28
29	SQ4, RP	(6, 18, 68)	1.88	0.28	29
FOR031.DAT	DC-10, SF	(15, 435, 439)	14.20	0.90	18
18	SQ4	(6, 52, 83)	2.24	0.22	19
FOR031.DAT	DC-5, SF	(12, 267, 280)	15.22	1.10	24
24	SQ4	(5, 11, 47)	0.74	0.18	25
FOR031.DAT	DC-1	(10, 358, 628)	7.44	0.50	10
10	SQ4	(10, 220, 548)	2.72	0.32	11

15 + 17 + 19	SF, SQ4, RP	(12, 90, 255)	2.16	0.22	
21 + 23 + 25	SF, SQ4	(8, 27, 109)	1.78	0.28	
27 + 29 + 11	SF, SQ4, RP	(7, 20, 88)	1.22	0.24	
15 + 17	SF, SQ4	(10, 214, 518)	1.10	0.22	
21 + 23	SF, SQ4	(12, 112, 329)	3.68	0.42	
27 + 29	SF, SQ4, RP	(6, 18, 68)	0.74	0.26	*1
15 + 19	SF, SQ4	(10, 78, 208)	1.52	0.14	
21 + 25	SF, SQ4, RP	(6, 18, 68)	0.56	0.18	*5
27 + 11	SF, SQ4	(11, 104, 289)	3.84	0.44	
PRESN+30+31	DC-10, SF	(20, 435, 542)	21.92	1.26	
	SQ4, RP	(7, 62, 109)	1.76	0.36	
PRESN+30+31	DC-5, SF	(16, 332, 414)	25.48	0.76	
	SQ4	(8, 65, 139)	1.90	0.38	
PRESN+30+31	DC-1	(16, 266, 398)	48.76	1.94	
	SQ4, RP	(8, 64, 129)	1.44	0.54	
PRESN+30	DC-10	(10, 359, 633)	2.82	0.20	
	SQ4	(10, 214, 518)	2.64	0.38	
PRESN+30	DC-5	(10, 359, 633)	3.76	0.22	
	SQ4	(10, 214, 518)	2.76	0.28	
PRESN+30	DC-1	(10, 484, 850)	6.78	0.62	
	SQ4	(10, 329, 733)	3.16	0.40	
PRESN+31	DC-10, SF	(19, 356, 510)	17.72	0.94	
	SQ4	(10, 88, 192)	1.86	0.28	
PRESN+31	DC-5, SF	(17, 306, 445)	19.36	1.00	
	SQ4	(9, 71, 163)	1.42	0.32	
PRESN+31	DC-1	(16, 370, 506)	36.22	1.72	
	SQ4	(10, 78, 212)	2.38	0.28	
14 + 16 + 18	SF, SQ4	(6, 18, 68)	11.98	0.38	*10
20 + 22 + 24	SF, SQ4	(6, 18, 68)	7.98	0.42	*5
26 + 28 + 10	SF, SQ4	(6, 18, 68)	7.26	0.40	*1

(iv) AC-2, AS-5

In this case we get the following from MODTC :

Index = 7 Max = 421 Total = 429

29 relations written to file PRESN.DAT

The maximum tree size is 542

File FOR030.DAT contains 194 relations

File FOR031.DAT contains 52 relations

Elapsed user time = 3.68 Elapsed system time = 0.02

Then from NTTRANS we obtain :

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time	Results File
PRESN.DAT	EH10	(9, 1742, 3472)	2.64	0.24	14
14	SQ4	(9, 1742, 3472)	9.68	0.42	15
PRESN.DAT	EH5	(9, 1742, 3472)	3.28	0.28	20
20	SQ4	(9, 1742, 3472)	9.98	0.44	21
PRESN.DAT	EH1	(9, 1742, 3472)	3.86	0.16	26
26	SQ4	(9, 1742, 3472)	9.66	0.12	27
FOR030.DAT	DC-10, SF	(10, 1263, 1934)	4.50	0.12	16
16	SQ4	(10, 1239, 1910)	10.28	0.54	17
FOR030.DAT	DC-5, SF	(10, 1263, 1934)	5.04	0.28	22
22	SQ4	(10, 1239, 1910)	10.14	0.48	23
FOR030.DAT	DC-1	(10, 1263, 1934)	9.62	0.28	28
29	SQ4	(10, 1239, 1910)	10.72	0.34	29
FOR031.DAT	DC-10, SF	(23, 626, 676)	28.42	0.72	18
18	SQ4	(5, 11, 47)	2.92	0.26	19
FOR031.DAT	DC-5, SF	(23, 736, 775)	31.96	1.50	24
24	SQ4	(6, 17, 71)	4.46	0.36	25
FOR031.DAT	DC-1	(22, 595, 623)	59.34	2.20	10
10	SQ4	(6, 18, 68)	2.30	0.16	11
15 + 17 + 19	SF, SQ4, RP	(8, 31, 113)	5.36	0.30	
21 + 23 + 25	SF, SQ4, RP	(8, 31, 113)	5.38	0.46	
27 + 29 + 11	SF, SQ4, RP	(8, 31, 113)	6.10	0.52	
15 + 17	SF, SQ4	(15, 2197, 3510)	27.34	0.48	
21 + 23	SF, SQ4	(15, 2197, 3510)	25.92	0.84	
27 + 29	SF, SQ4	(15, 2197, 3510)	27.88	0.20	

15 + 19	SF, SQ4, RP	(7, 25, 110)	9.98	0.42	
21 + 25	SF, SQ4	(8, 31, 113)	1.24	0.32	
27 + 11	SF, SQ4, RP	(8, 40, 160)	2.40	0.20	
PRESN+30+31	DC-10, SF	(28, 1002, 1183)	44.52	1.84	
	SQ4, RP	(7, 23, 93)	5.94	0.56	
PRESN+30+31	DC-5, SF	(21, 382, 482)	51.56	1.36	
	SQ4	(7, 25, 89)	1.22	0.20	*5
PRESN+30+31	DC-1	(25, 585, 762)	98.70	3.42	
	SQ4	(7, 23, 93)	2.60	0.30	
PRESN+30	DC-10, SF	(10, 983, 1568)	4.64	0.40	
	SQ4	(10, 983, 1568)	6.24	0.52	
PRESN+30	DC-5, SF	(11, 979, 1556)	5.40	0.12	
	SQ4	(11, 979, 1556)	6.34	0.24	
PRESN+30	DC-1	(11, 932, 1513)	8.50	0.26	
	SQ4	(11, 928, 1505)	8.02	0.42	
PRESN+31	DC-10, SF	(31, 907, 1119)	37.04	2.04	
	SQ4	(6, 18, 68)	5.22	0.34	*10
PRESN+31	DC-5, SF	(27, 658, 783)	38.78	1.38	
	SQ4	(8, 30, 139)	2.76	0.20	
PRESN+31	DC-1	(26, 539, 713)	73.90	2.64	
	SQ4, RP	(7, 23, 93)	2.94	0.32	
14 + 16 + 18	SF, SQ4, RP	(8, 35, 109)	50.46	1.04	
20 + 22 + 24	SF, SQ4, RP	(8, 30, 114)	54.44	0.74	
26 + 28 + 10	SF, SQ4, RP	(7, 25, 89)	33.86	3.10	*1

The smallest triplet for a presentation obtained, (6, 18, 68), crops up in a number of places, at least once in each of the four tables. One such presentation is :

$$\langle x, y \mid x^2 = y^4 = (xy^{-1})^5 = (xy^{-2})^5 = (xyxy^{-1})^4 = (xyxy^{-2}xy^{-1})^3 = 1 \rangle$$

from (ii). Compare this to the presentation (4, 7, 31) given in [16] which is :

$$\langle x, y \mid x^2 = y^4 = (xy)^5 = ((xy)^2)^5 = 1 \rangle.$$

From these tables there are a number of points to which it is worth drawing attention :

(a) With MODTC the Felsch enumerations take longer to carry out than the HLT ones.

(b) The HLT methods also give better presentations from PRESN.DAT alone than the Felsch ones do.

(c) As might be expected, the more frequent the substring searching the longer it takes, in general, for each file to be decoded. [This is not always the case. Sometimes, a more frequent substring search will substantially reduce the size of the presentation at a critical point, then subsequent eliminations and substring searches will take far less time to execute, even though the substring searches are frequent.]

(d) Even although FOR030.DAT contains no interesting relators in the HLT cases, adding FOR030.DAT to PRESN.DAT before decoding with DC-(n) gives a presentation of shorter length than decoding PRESN.DAT itself with EH(n). [Perhaps I should have tried decoding PRESN.DAT alone with DC-(n) for comparison ?]

No set method of decoding and combination of the different sets of relations gives the best presentation in each case. However, in each of (i), (ii), (iii) and (iv) the extra relations have to be used to find short presentations.

Example 5

Now I will look at an example without the "extra relations" and will demonstrate the use of the commands AE and AD. From [12] I chose an example with a much larger index, namely the enumeration of PSL(2,11) in the group J_1. This has an index of 266 and I decided to use the first presentation, 15.1. This is given by :

$$G = J_1 = \langle a, b \mid a^2 = b^3 = (ab)^7 = [a, b]^{10} = [a, b^{-1}(ab)^2]^6 = 1 \rangle,$$

$$H = \langle a, b^{ab(ab^{-1})^2} \rangle = PSL(2,11).$$

My first attempt to carry out the enumeration with MODTC was using the HLT method with parameters AC-4, AM20000 and MT1. This yielded the following results :

Index = 266 Max = 20000 Total = 24191
1055 relations written to file PRESN.DAT
The maximum tree size is 24536

Elapsed user time = 170.26 Elapsed system time = 1.94

The default tree size is 10001, so in this case the tree was increased automatically in the middle of the enumeration and the calculation restarted from scratch. This accounts for the length of time required to complete the enumeration. The run was repeated with AT20000 and this time the elapsed user and system times were 92.06s and 2.60s respectively.

I then tried the same enumeration with a smaller value of MAXTAB, namely 15000. This time I altered the tree size to 20000 before enumerating, and I obtained :

```
Index = 266    Max = 15000  Total = 18394
1055 relations written to file PRESN.DAT
The maximum tree size is 18440

Elapsed user time = 69.54      Elapsed system time = 0.50
```

The enumeration was then repeated for values of MAXTAB of 10000, 12000, 13000 and 14000 and only in the latter two cases did it terminate successfully. This time the results were :

AM13000, AT20000

```
Index = 266    Max = 13000  Total = 16144
1055 relations written to file PRESN.DAT
The maximum tree size is 15786

Elapsed user time = 80.58      Elapsed system time = 0.42
```

AM14000, AT20000

```
Index = 266    Max = 14000  Total = 17241
1055 relations written to file PRESN.DAT
The maximum tree size is 17128

Elapsed user time = 83.00      Elapsed system time = 1.02
```

I then repeated the example with a MAXTAB value of 40000. In this case I used AT50000 and obtained the following results :

Index = 266 Max = 40000 Total = 43980

1055 relations written to file PRESN.DAT

The maximum tree size is 39804

Elapsed user time = 57.14 Elapsed system time = 5.76

From these it appears that the larger the MAXTAB given, the greater the final size of the tree. In fact, the tree size always seems to be within 5000 of the value of the MAXTAB, or alternatively, (apart from the last set of results), is very close - within 500 - of the total number of cosets defined !

I chose two of these runs to study in more detail, namely (i) AM15000 and (ii) AM13000.

(i) AC-4, AM15000

Here I decoded PRESN.DAT using the command EH10. This used up all of the available space in the array GEN, so did not complete. I then repeated this using AE10 instead, with NTIMES=20. This gave the following results :

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time
PRESN.DAT	AE10	(49, 47596, 59851)	4379.40	44.82
	SQ4	(49, 46940, 59159)	2901.92	60.96

In the decoding phase a total of 365 eliminations were carried out.

(ii) AC-4, AM13000

Here I decoded PRESN.DAT using the commands EH10 and AE10 with NTIMES=20. This gave the following results :

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time
PRESN.DAT	EH10	(47, 55284, 72661)	3765.54	20.40
	SQ4	(47, 54910, 72279)	2104.70	1.74
PRESN.DAT	AE10	(47, 41530, 53557)	4089.46	17.90
	SQ4	(47, 39592, 51571)	2864.64	2.40

In the decoding phase a total of 370 eliminations were carried out in the first of these runs, 356 in the second.

I ran MODTC again with the Felsch method using strategies AS5, AS1 and AS-5 in turn. In each case the other parameters were set with the following commands - AC-2, AM20000, MT1. The results obtained were :

(iii) AC-2, AS5

From MODTC :

Index = 266 Max = 6031 Total = 6108
801 relations written to file PRESN.DAT
The maximum tree size is 2642

Elapsed user time = 220.64 Elapsed system time = 2.10

These times were obtained for the enumeration using a file FOR003.DAT where no relations (except for involutions) were given exponents > 1. I then repeated the example with a new version of FOR003.DAT in which all relators were written as a subword raised to a power if possible. This resulted in an elapsed user time of 36.58 seconds and an elapsed system time of 0.32 seconds - quite an improvement on the original times. This new FOR003.DAT was used for the rest of the Felsch runs of this example.

Processing PRESN.DAT with NTTRANS I obtained the following results :

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time
PRESN.DAT	EH10	(49, 3069, 3981)	109.46	6.60
	SQ4, RP	(14, 116, 379)	38.12	2.96
PRESN.DAT	AN20, AE10	(49, 2971, 3879)	153.12	1.72
	SQ4, RP	(12, 110, 349)	24.10	0.46

In both of these decoding phases, 43 eliminations were carried out.

(iv) AC-2, AS1

Here the following results were obtained from MODTC :

Index = 266 Max = 6243 Total = 6266
800 relations written to file PRESN.DAT
The maximum tree size is 2714

Elapsed user time = 36.88 Elapsed system time = 0.30

Then, from NTTRANS :

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time
PRESN.DAT	EH10	(50, 3669, 4803)	95.04	0.82
	SQ4, RP	(12, 116, 343)	107.64	0.96
PRESN.DAT	AN20, AE10	(50, 3281, 4305)	105.72	1.34
	SQ4, RP	(13, 120, 399)	82.00	0.80

In both of these, 41 eliminations were carried out.

(v) AC-2, AS-5

From MODTC I obtained :

Index = 266 Max = 5414 Total = 5447

852 relations written to file PRESN.DAT

The maximum tree size is 2396

Elapsed user time = 32.12 Elapsed system time = 0.24

Then, from NTTRANS :

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time
PRESN.DAT	EH10	(49, 68950, 87443)	717.12	16.66
	SQ4	(49, 68374, 86815)	3269.40	19.16
PRESN.DAT	AN20, AE10	(50, 19506, 24505)	354.92	3.34
	SQ4	(50, 19216, 24179)	1143.28	6.88

In the first case 73 eliminations were carried out, in the second 78.

From these results it is clear that the Felsch enumeration and the subsequent processing of the results with NTTRANS is far more successful than the corresponding HLT ones. The Felsch enumerations, themselves, are quicker (if exponents are given) and there are not so many high-numbered generators that have to be eliminated. [Note that the number of high-numbered generators to be eliminated in the decoding of PRESN.DAT is not always proportional to the size of the tree.]

With the HLT examples, ((i) and (ii)), the AE command has to be used to prevent the relators themselves from becoming too long. In the Felsch examples ((iii), (iv) and (iv)), only the last one gives a substantially better result with AE10, and

this is also the only one in which AE10 is quicker to execute than EH10.

One of the best presentations found - that of (12, 110, 349) from (iii) is as follows :

$$x^2 = 1$$
$$y^3 = 1$$
$$(xyxy^{-1})^5 = 1$$
$$(xy^{-1})^{11} = 1$$
$$((xy)^3 (xy^{-1})^3)^2 = 1$$
$$(xy)^3 xy^{-1} (xy)^3 xy^{-1} xy(xy^{-1})^2 xyxy^{-1} = 1$$
$$((xy)^2 (xy^{-1})^2 xy(xy^{-1})^2)^2 = 1$$
$$((xyxy^{-1})^2 xy^{-1})^3 = 1$$
$$((xy)^2 (xy^{-1})^4)^3 = 1$$
$$((xy)^2 (xy^{-1})^2)^6 = 1$$
$$((xy)^2 (xy^{-1})^3)^5 = 1$$
$$(xy(xy^{-1})^4)^6 = 1$$

This is compared to that given in [12] which is due to Sunday :

$$\langle S, T \mid S^{11} = T^2 = (ST)^3 = (S^6 TS^4 T)^2 = 1 \rangle \text{ where } S=xy \text{ and } T=x$$
$$\cong \langle x, y \mid (xy)^{11} = x^2 = (x^2 y)^3 = ((xy)^6 x(xy)^4 x)^2 = 1 \rangle$$
$$\cong \langle x, y \mid (xy^{-1})^{11} = x^2 = y^3 = (y(xy)^5 y(xy)^3)^2 = 1 \rangle.$$

Now I will look at a couple of very large examples and show what can be achieved with the aid of the extra relations. Due to the time needed to decode and simplify these presentations, I will not attempt such a variety of methods as I did with earlier examples.

Example 6

The first large example I would like to look at is presentation 8.1 from [16], namely :

$$G = A_8 = \langle\, a, b \mid a^2 = b^4 = (ab)^{15} = (ab^2)^4 = (ab)^5\, a\, b^2\, ab(ab^{-1})^2\, (ab)^2\, ab^{-1}\, (ab)^7\, ab^{-1} = 1 \,\rangle,$$

$$H = \langle\, b^2, (ab)^3\, ab^{-1}\, a \,\rangle \cong A_7 .$$

Firstly I tried this on MODTC as a straight HLT with parameters MT-1 and AC-4 to get the following statistics :

Index = 8 Max = 4392 Total = 4537
34 relations written to file PRESN.DAT
The maximum tree size is 6020
File FOR030.DAT contains 33 relations
File FOR031.DAT contains 440 relations

Elapsed user time = 2.88 Elapsed system time = 0.16

I then repeated this with a MAXTAB of 1800 to get :

Index = 8 Max = 1800 Total = 1987
34 relations written to file PRESN.DAT
The maximum tree size is 2946
File FOR030.DAT contains 189 relations
File FOR031.DAT contains 239 relations

Elapsed user time = 3.72 Elapsed system time = 0.20

I decided that this looked the better of the two runs to use since the tree was smaller. From NTTRANS I obtained :

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time	Results File
PRESN.DAT	EH10	(16, 27539, 34852)	273.38	2.28	14
14	SQ4	(16, 27539, 34852)	115.52	0.94	15
FOR030.DAT	DC-10	(2, 2, 6)	3.88	0.40	16
FOR031.DAT	DC-10, SF	(80, 3124, 3279)	3719.26	15.06	17
17	SQ4	(14, 118, 298)	81.78	0.64	18
15 + 16 + 18	SF, SQ4	(17, 147, 442)	1004.88	4.62	19

This presentation FOR019.DAT is as follows :

$x^2 = 1$

$y^4 = 1$

$(xy^{-1})^7 = 1$

$(xy^{-2})^6 = 1$

$(xyxy^{-1})^5 = 1$

$(xyxy^{-2})^4 = 1$

$(xyxy^{-1}xy^{-2})^3 = 1$

$(xyxy^{-2}xy^{-1})^3 = 1$

$((xy)^2(xy^{-1})^2)^3 = 1$

$((xy)^2xy^{-1}(xy)^2xy^{-2})^2 = 1$

$((xy)^2xy^{-2})^4 = 1$

$((xyxy^{-1})^2xy^{-1})^3 = 1$

$(xyxy^{-2}xy^{-1}xy^{-2})^3 = 1$

$(xy)^3xy^{-1}(xy)^2xy^{-1}(xy)^3xy^{-1}xy(x^{-1}y^2)^2x^{-1}yxy^{-1} = 1$

$(xyxy^{-1}(xy^{-2})^2)^4 = 1$

$((xy)^3(xy^{-1})^2)^5 = 1$

$((xyxy^{-1})^2xy^2x^{-1}y^{-1}x^{-1}y)^4 = 1$

This is compared to the one given in [12], presentation 4.1 :

$$\langle x, y \mid x^2 = y^4 = (xy)^7 = [x, y]^5 = (xyxy^2xy^{-1})^3 = 1 \rangle$$

which is (5, 15, 61).

These are just the 1st, 2nd, 3rd, 5th and 8th relations of FOR019.DAT in a slightly disguised form. From the structure of the other relations, this looks as if it may be another example in which the fact that only the subwords of exponentiated relators are searched for matching substrings hinders the simplification.

I repeated this example using the commands AE and AD to decode, but did not get such good results.

Example 7

Another large example which has proved very difficult to do is the enumeration of the subgroup of index 40 in the group $Sp(4,3) \cong PSU(4,2)$. The presentation for Sp(4,3) and the subgroup generators are are follows :

$$Sp(4,3) = \langle a, b \mid a^2 = b^4 = (ab)^9 = [a, b]^5 = 1,$$
$$a b^{-1} a b^2 a b^{-1} (ab)^2 a b^2 a b a b^2 (ab^{-1})^2 (ab)^4 = 1 \rangle,$$

$$H = \langle b^2, (ab^{-1})^2 ab(ba)^3, a b^{-1} ab(ab^2)^2 (ab^{-1})^2 ab \rangle.$$

This presentation for Sp(4,3) is 10.1 in [16], but these particular generators for H were given to me by A. Jamali, a fellow research student, as an example he had found difficult to do using the Reidemeister-Schreier process.

When this enumeration is attempted as an HLT with no MAXTAB, the following statistics are obtained :

Index = 40 Max = 2775 Total = 3567
169 relations written to file PRESN.DAT
The maximum tree size is 4540
File FOR030.DAT contains 58 relations
File FOR031.DAT contains 325 relations

Elapsed user time = 2.54 Elapsed system time = 0.10

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time	Results File
PRESN.DAT	EH10	(41, 23081, 26641)	360.50	4.38	14
14	SQ4	(41, 17675, 19244)	2279.08	39.34	15
FOR030.DAT	DC-10, SF	(7, 649, 875)	29.60	2.98	16
16	SQ4	(7, 649, 875)	2.88	0.16	17
FOR031.DAT	DC-10, SF	(128, 5122, 5302)	11535.38	60.48	18
18	SQ4	(12, 72, 125)	168.84	0.40	19
15 + 17 + 19	SF	(56, 9961, 10973)	43.12	4.64	
	SQ4	(18, 135, 341)	744.82	74.04	20
15 + 19	SF	(51, 9636, 10520)	27.14	0.10	
	SQ4	(15, 99, 224)	715.68	14.24	11

I decided that this example was a prime candidate for the AE and AD commands. However, with n=10 and NTIMES=50, I did not have much success.

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time	Results File
PRESN.DAT	AE10	(41, 21347, 24579)	457.02	2.60	
	SQ4	(41, 17594, 19226)	110.52	0.56	14
FOR030.DAT	AD-10, SF	(7, 649, 875)	29.32	0.96	15
15	SQ4	no change			
FOR031.DAT	AD-10, SF	(128, 6303, 6460)	15539.08	40.58	
	SQ4	(127, 5277, 5429)	1178.14	5.74	17
14 + 15 + 17	SF	(171, 22048, 24018)	321.76	3.34	
	SQ4	(171, 18051, 19788)	7639.50	41.16	

Taking selected relations from FOR017.DAT and FOR015.DAT and adding them to FOR014.DAT before applying SQ4 produced no significantly better results.

I also tried this example with MAXTAB=280. This time I obtained the following statistics from NTTRANS :

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time	Results File
PRESN.DAT	EH10	(42, 6664, 7561)	163.18	0.90	
	SQ4	(42, 6351, 7277)	256.48	1.08	21
FOR030.DAT	DC-10, SF	(43, 2755, 3178)	119.46	0.44	
	SQ4	(43, 2316, 2671)	101.74	0.84	23
FOR031.DAT	DC-10, SF	(17, 1018, 1125)	30.20	0.92	
	SQ4	(17, 1012, 1119)	13.40	0.46	25
21 + 23 + 25	SF, SQ4	(95, 7396, 8290)			

Again I tried different combinations of relations but the best presentation I found was that in FOR021.DAT. However, if I added FOR019.DAT (from the previous page) to FOR021.DAT I got (14, 63, 182). This was the best presentation found by any method.

Repeating this example with AE10 and AD-10, the best presentation I could find was (55, 5696, 6641).

I then tried the same example with a different value of MAXTAB, namely 500. This time my best presentation was (55, 4544, 5404). When finally I tried a Felsch enumeration with STOPLP=+5 I could get no better than (57, 8000, 8945).

The fact that I obtained such a good presentation with the first method was, that by some lucky accident, I hit on some nice short relations in FOR019.DAT. The relations written to FOR019.DAT are :

$x^2 = 1$

$y^2 = 1$

$yzyz^{-1} = 1$

$(xy^{-1})^3 = 1$

$z^6 = 1$

$(xz)^4 = 1$

$(xzxz^{-1})^2 = 1$

$(yz^{-3})^2 = 1$

$(xy^{-1}z^{-3})^3 = 1$

$xy^{-1}xzx^{-1}yzx^{-1}yzx^{-1}z^{-1}yx^{-1}z^{-1}yx^{-1}z^{-1} = 1$

$xy^{-1}xz^{-1}x^{-1}yzx^{-1}yzx^{-1}z^{-2}yx^{-1}z^{-2}yx^{-1}z^2 = 1$

$(xy^{-1}xzx^{-1}yzx^{-1}z^{-1})^3 = 1$

The third of these relations which, along with the first, implies that $[y, z]=1$, is especially useful.

For comparison, the best presentation I obtained by any method was :

$x^2 = 1$

$y^2 = 1$

$yzyz^{-1} = 1$

$(xy^{-1})^3 = 1$

$z^6 = 1$

$(xz)^4 = 1$

$(xzxz^{-1})^2 = 1$

$(yz^{-3})^2 = 1$

$(xy^{-1}z^{-3})^3 = 1$

$(xzy^{-1}z^2)^3 = 1$

$xy^{-1}xzx^{-1}yzx^{-1}z^{-1}yx^{-1}z^{-1}yx^{-1}z^{-1}yx^{-1}z = 1$

$(xy^{-1}z)^9 = 1$

$(xy^{-1}zxz^{-1}yx^{-1}yz)^3 = 1$

$(xy^{-1}z^2)^9 = 1$

Finally, I will demonstrate the use of the command OD to start an enumeration from a coset table in which INDEX cosets are already defined. In most of the cases I have looked at, the total number of cosets that have to be defined increases, but the size of the tree decreases. I have chosen examples with a large index as those will have more of an effect on the statistics obtained.

(i) First of all I will look at J_1 from Example 5. The results on the left are those obtained from MODTC, enumerating from scratch. The results on the right are obtained using OD afterwards. In each case MT1 is used.

AC-4, AM40000, AT50000

Ind = 266 Max = 40000 Total = 43980	Ind = 266 Max = 40000 Total = 44552
1055 relations written to PRESN.DAT	844 relations written to PRESN.DAT
The maximum tree size is 39804	The maximum tree size is 33202
User time = 57.14 System time = 5.76	User time = 53.52 System time = 0.60

AC-4, AM20000, AT50000

Ind = 266 Max = 20000 Total = 24191	Ind = 266 Max = 20000 Total = 24862
1055 relations written to PRESN.DAT	844 relations written to PRESN.DAT
The maximum tree size is 24536	The maximum tree size is 18744
User time = 85.02 System time = 0.96	User time = 79.74 System time = 1.06

AC-4, AM15000, AT20000

Ind = 266 Max = 15000 Total = 18394	
1055 relations written to PRESN.DAT	Does not complete - runs out of space.
The maximum tree size is 18440	
User time = 69.54 System time = 0.50	

AC-4, AM13000, AT20000

Ind = 266 Max = 13000 Total = 16144	
1055 relations written to PRESN.DAT	Does not complete - runs out of space.
The maximum tree size is 15786	
User time = 80.58 System time = 0.42	

AC-2, AS5

Ind = 266 Max = 6031 Total = 6108
801 relations written to PRESN.DAT
The maximum tree size is 2642
User time = 36.58 System time = 0.32

Ind = 266 Max = 5830 Total = 5905
800 relations written to PRESN.DAT
The maximum tree size is 2460
User time = 34.82 System time = 0.42

AC-2, AS1

Ind = 266 Max = 6243 Total = 6266
800 relations written to PRESN.DAT
The maximum tree size is 2714
User time = 36.88 System time = 0.30

Ind = 266 Max = 7708 Total = 7824
801 relations written to PRESN.DAT
The maximum tree size is 3700
User time = 43.92 System time = 0.62

AC-2, AS-5

Ind = 266 Max = 5414 Total = 5447
852 relations written to PRESN.DAT
The maximum tree size is 2396
User time = 32.90 System time = 0.72

Ind = 12848 Max =12848 Total = 12949
813 relations written to PRESN.DAT
The maximum tree size is 5908
User time = 68.84 System time = 0.30

From this set of results it can be seen that, in general, the enumeration with the OD command takes less time to complete than the original one. However in these enumerations which completed with OD under the specified MAXTAB restriction, fewer relations were written to PRESN.DAT and a smaller tree was built up. In all of the HLT enumerations more cosets had to be defined in the OD run. This will nearly always be the case. Since the "original" definitions may be spread at random throughout the table, a larger total number of cosets have to be defined during the enumeration, to fill up the gaps between the definitions in the first few rows of the subgroup generator and relator tables, before deductions are found.

In contrast, the Felsch enumerations, apart from the AS5 case, appear far worse with the OD option, both in terms of the tree size and the time taken for the enumeration.

(ii) The second example I have chosen to look at is the presentation for $B_{2,4}$ denoted by G_2 in [19] :

$$G = B_{2,4} = \langle\ a, b \mid a^4 = b^4 = (ab)^4 = (a^{-1}b)^4 = (a^2b)^4 = (ab^2)^4 = (a^2b^2)^4 = [a,b]^4 = (a^{-1}bab)^4 = 1\ \rangle,$$

$$H = \langle\ a\ ,\ b^2\ \rangle.$$

This subgroup H has index 64 in $B_{2,4}$.

AC-4, (no MAXTAB)

Ind = 64 Max = 157 Total = 204
574 relations written to PRESN.DAT
The maximum tree size is 474
User time = 1.12 System time = 0.10

Ind = 64 Max = 177 Total = 222
558 relations written to PRESN.DAT
The maximum tree size is 450
User time = 1.08 System time = 0.06

AC-4, AM100

Ind = 64 Max = 100 Total = 116
570 relations written to PRESN.DAT
The maximum tree size is 392
User time = 1.28 System time = 0.08

Ind = 64 Max = 100 Total = 102
546 relations written to PRESN.DAT
The maximum tree size is 276
User time = 1.26 System time = 0.02

AC-4, AM70

Ind = 64 Max = 70 Total = 104
566 relations written to PRESN.DAT
The maximum tree size is 358
User time = 1.36 System time = 0.08

Does not complete - runs out of space.

AC-2, AS5

Ind = 64 Max = 64 Total = 73
538 relations written to PRESN.DAT
The maximum tree size is 152
User time = 2.38 System time = 0.10

Ind = 64 Max = 65 Total = 67
570 relations written to PRESN.DAT
The maximum tree size is 326
User time = 2.06 System time = 0.10

AC-2, AS1

Ind = 64 Max = 73 Total = 86
546 relations written to PRESN.DAT
The maximum tree size is 194
User time = 2.78 System time = 0.00

Ind = 64 Max = 67 Total = 71
562 relations written to PRESN.DAT
The maximum tree size is 320
User time = 2.18 System time = 0.00

AC-2, AS-5

Ind = 64 Max = 64 Total = 67
526 relations written to PRESN.DAT
The maximum tree size is 154
User time = 2.42 System time = 0.06

Ind = 64 Max = 79 Total = 84
554 relations written to PRESN.DAT
The maximum tree size is 300
User time = 2.54 System time = 0.04

Again the HLT enumerations show an improvement in statistics with OD if the value of MAXTAB is large enough to allow the enumeration to complete. This time two of the Felsch enumerations show a decrease in the total number of cosets defined; however, the size of the tree and the number of relations written to PRESN.DAT increases in all of the Felsch runs.

(iii) For my third example I will take Sidki's group Y(3,6) from [33] given by :

$$G = Y(3,6) = \langle\ a, b, c \mid a^6 = b^6 = c^6 = (ab)^2 = (ac)^2 = (bc)^2 = (a^2 b^2)^2 = (a^2 c^2)^2 = (b^2 c^2)^2 = (a^3 b^3)^2 = (a^3 c^3)^2 = (b^3 c^3)^2 = 1\ \rangle,$$

$$H = \langle\ a\ ,\ b\ \rangle.$$

AC-4, (no MAXTAB)

Ind = 160 Max = 343 Total = 449
1914 relations written to PRESN.DAT
The maximum tree size is 1830
User time = 3.44 System time = 0.06

Ind = 160 Max = 368 Total = 494
1834 relations written to PRESN.DAT
The maximum tree size is 1668
User time = 3.12 System time = 0.10

AC-4, AM300

Ind = 160 Max = 300 Total = 364
1914 relations written to PRESN.DAT
The maximum tree size is 1436
User time = 3.82 System time = 0.04

Ind = 160 Max = 300 Total = 345
1830 relations written to PRESN.DAT
The maximum tree size is 1048
User time = 3.22 System time = 0.06

AC-4, AM200

Ind = 160 Max = 200 Total = 309
1914 relations written to PRESN.DAT
The maximum tree size is 1482
User time = 3.46 System time = 0.18

Ind = 160 Max =200 Total = 211
1826 relations written to PRESN.DAT
The maximum tree size is 796
User time = 3.16 System time = 0.10

AC-4, AM170

Ind = 160 Max = 170 Total = 266
1914 relations written to PRESN.DAT
The maximum tree size is 1222
User time = 4.12 System time = 0.00

Does not complete - runs out of space.

AC-2, AS5

Ind = 160 Max = 160 Total = 163
1816 relations written to PRESN.DAT
The maximum tree size is 590
User time = 3.64 System time = 0.06

Ind = 160 Max =161 Total = 161
1900 relations written to PRESN.DAT
The maximum tree size is 1262
User time = 3.68 System time = 0.04

AC-2, AS1

Ind = 160 Max = 165 Total = 193
1830 relations written to PRESN.DAT
The maximum tree size is 700
User time = 3.76 System time = 0.08

Ind = 160 Max =184 Total = 193
1878 relations written to PRESN.DAT
The maximum tree size is 1250
User time = 3.76 System time = 0.10

AC-2, AS-5

Ind = 160 Max = 160 Total = 194
1844 relations written to PRESN.DAT
The maximum tree size is 680
User time = 3.84 System time = 0.06

Ind = 160 Max =161 Total = 161
1898 relations written to PRESN.DAT
The maximum tree size is 1236
User time = 3.74 System time = 0.10

With the three Felsch enumerations using OD, fewer (or the same number of) cosets are defined in total, however the tree is much larger in each case.

With this example the HLT enumerations are again quicker with OD and produce a smaller tree. This time, in two out of the three runs which complete, fewer cosets need to be defined in total.

In fact, using the results from the AC-4, AM200 case with NTTRANS, we find the following :

Original enumeration

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time	Results File
PRESN.DAT	EH10	(27, 173, 362)	678.14	2.20	16
16	SQ4	(14, 67, 162)	0.72	0.22	17

209 eliminations were carried out.

OD enumeration

Input File	Command(s)	Relations	Elapsed User Time	Elapsed System Time	Results File
PRESN.DAT	EH10	(21, 122, 248)	489.56	1.80	14
14	SQ4	(11, 38, 108)	0.78	0.18	15

185 eliminations were carried out.

Thus, the presentation obtained from OD is very much better in this case - it gives a shorter final presentation in a much quicker time.

In general the usefulness of the OD command depends on the example in question. In some cases it will give a better presentation with certain parameters, in others it won't. However, for an HLT enumeration with a large index it certainly seems to be worth trying, especially if the tree turns out to be very large.

Conclusion

None of the techniques discussed in this chapter to carry out the modified Todd-Coxeter algorithm, or simplify the resulting presentation, works well in all cases. In some examples, HLT enumerations give the best results, in others, Felsch. The "extra relations" do seem to be very useful in many cases where PRESN.DAT is decoded down to a very long final presentation. Similarly, the OD command appears to give smaller trees in HLT runs of examples with a fairly large index.

With such a process as the modified algorithm, the more methods that are available to the user, the greater the chance of finding a reasonable result.

Chapter 5

The Groups $\bar{N}(\beta, n)$

In this chapter I shall look at a certain family of presentations and try to determine the structure of the groups concerned. In the first section I will show where these presentations originated, and in the second I will derive certain general results about the corresponding groups. Then, in the last two sections, I will examine the structure of these groups for specific small values of the parameters β and n, with the aid of the programs described in the previous chapters.

§5.1 The Origin of the Groups $\bar{N}(\beta, n)$

When dealing with group presentations, one of the most important questions to ask is whether the corresponding group is finite or infinite. If the group is finite we can usually determine the order and structure of the group without much difficulty. However, if it is infinite, this is not such an easy matter. One interesting type of group presentation is a one relator product of cyclic groups. In general, a one-relator product of the groups $\{ A_i : i \in I \}$ is a quotient $(*A_i) / N(R)$ where :

$(*A_i)$ is the free product of the groups A_i $(i \in I)$,
R is a cyclically reduced word, and
N(R) is the normal closure of R in $(*A_i)$.

The case where the A_i are finite cyclic groups is of particular interest, especially when $|I| = 2$. In this case, if A_1 and A_2 are cyclic of orders m and n respectively, we have a presentation of the form

$$\langle a, b \mid a^m = b^n = R(a, b) = 1 \rangle$$

where $m > 0$, $n > 0$, and R is a word of the form a^{i_1}, b^{j_1} or $a^{i_1} b^{j_1} \ldots . a^{i_r} b^{j_r}$, with $r \geq 1$, $0 < i_p < m$ for all p, and $0 < j_q < n$ for all q.

Groups of this type have been investigated by E.F. Robertson, C.M. Campbell and R.M. Thomas over the last few years, and their structure determined for various relators, R. Some of their findings have been published in

[13] and [14]. M.D.E. Conder also considered groups of this type and he determined which of the presentations of the form

$$\langle a, b \mid a^2 = b^3 = R(a, b) = 1 \rangle$$

define finite groups where the relator R has length at most 24 [17].

I became involved with the investigation of one such presentation, namely that of the group $G(\alpha, n)$ defined by :

$$\langle a, b \mid a^2 = b^n = ab^{-1}ab(abab^{-1})^{\alpha-1}ab^2ab^{-2} = 1 \rangle \qquad (1)$$

where $n \geq 1$ and $\alpha \geq 1$. Many results discovered about these groups appear in the University of Leicester Technical Report [7] and papers [8], [9] and [10]. In section 5 of this report the following proposition is given :

Theorem 1 ([7], Prop. 5.2)

$G(\alpha, n)$ is finite if and only if the group $\bar{N}$ with presentation

$$\langle \beta, \tau \mid \beta^n = \tau^n = (\beta\tau^{-1})^{\alpha-1} = 1, \tau\beta^2 = \beta\tau^2 \rangle$$

is finite.

Proof

Let $c=aba$ and N be the subgroup $\langle b, c \rangle$ of $G(\alpha, n)$. Carrying out the modified Todd-Coxeter algorithm, $1b = b.1$ from the first subgroup generator, and filling this into all of the tables we get the following :

subgroup generator tables

b
1 | 1

a	b	a
1		1

augmented coset table

	a	b
1		b.1

relator tables

a	a
1	1

b	...	b		
1	1	...	1	1

a	b^{-1}	a	b	(a	b	a	$b^{-1})^{\alpha-1}$	a	b	b	a	b^{-1}	b^{-1}			
1												1	1	1		

Now define $1a = 2$ in the first group relator. This gives us $2a = 1$. When these two pieces of information are filled into the second subgroup generator table we get the deduction $2b = 2$.

Thus, $1\,a\,b\,a = c.1$

$\Rightarrow 2\,b\,a = c.1$

$\Rightarrow 2\,b = c.1a^{-1}$

$\Rightarrow 2\,b = c.2$.

This gives us enough information to completely close all of the tables. The fact that the values of n and α are unknown does not matter in this case. The first row of the second relator consists entirely of 1's, and the second row 2's, so it is irrelevant how many times the generator b occurs in the relator. In the third relator, $1abab^{-1} = 1$ and $2abab^{-1} = 2$, so again this subword can occur any number of times without affecting the rest of the entries in the row.

b
1 | 1

a	b	a
1 | 2	2	1

	a	b
1	2	b.1
2	1	c.2

a	a
1 | 2	1
2 | 1	2

b	...	b
1 | 1	...	1 | 1
2 | 2	...	2 | 2

a	b^{-1}	a	b	(a	b	a	b^{-1})$^{\alpha-1}$	a	b	b	a	b^{-1}	b^{-1}
1	2	2	1	1	2	2	1	1	2	2	2	1	1	1
2	1	1	2	2	1	1	2	2	1	1	1	2	2	2

Since $|G:N| = 2$, N is a normal subgroup of G.

We then trace out the subgroup relations :

Relations from subgroup generators

$1\,b = b.1 \Rightarrow b.1 = b.1 \Rightarrow b = b$ a trivial relation

$1\,a\,b\,a = c.1 \Rightarrow c.2\,a = c.1 \Rightarrow c.2 = c.1\,a^{-1} \Rightarrow c.2 = c.2 \Rightarrow c = c$ a trivial relation.

Relations from group relators

$a^2 = 1$ gives trivial relations only.

$b^n = 1$ gives the relations $b^n = 1$ and $c^n = 1$.

The last relator gives us the two relations :

$$c^{-1}b(cb^{-1})^{\alpha-1}c^2b^{-2} = 1 \text{ and } b^{-1}c(bc^{-1})^{\alpha-1}b^2c^{-2} = 1.$$

Thus, N has presentation :

$$\langle\ b, c\ \mid\ b^n = c^n = c^{-1}b(cb^{-1})^{\alpha-1}c^2b^{-2} = b^{-1}c(bc^{-1})^{\alpha-1}b^2c^{-2} = 1\ \rangle.$$

We now perform a sequence of Tietze transformations to simplify the last two relations. First of all we introduce the new generator $e = cb^{-1}$ and delete c, using $c = eb$, to obtain the following presentation :

$$\langle\ b, e\ \mid\ b^n = (eb)^n = 1, e^{\alpha-1} = b^{-1}eb^2e^{-1}b^{-1}e^{-1} = be^{-1}b^{-1}e^{-1}b^{-1}eb\ \rangle.$$

Now we introduce $d = beb^{-1}$ to get :

$$\langle\ b, d, e\ \mid\ b^n = (eb)^n = 1, b^{-1}db = e, b^{-1}eb = e^{\alpha}d, b^{-1}ebd^{-1}e^{-1} = d^{-1}e^{-1}b^{-1}eb\ \rangle.$$

From the last relation, $b^{-1}ebd^{-1}e^{-1} = d^{-1}e^{-1}b^{-1}eb$

$$\Rightarrow\quad e^{\alpha}dd^{-1}e^{-1} = d^{-1}e^{-1}e^{\alpha}d \qquad \text{using } b^{-1}eb = e^{\alpha}d$$

$$\Rightarrow\quad e^{\alpha-1} = d^{-1}e^{\alpha-1}d$$

$$\Rightarrow\quad [\,d, e^{\alpha-1}\,] = 1.$$

Thus, $N \cong \langle\ b, d, e\ \mid\ b^n = (eb)^n = [\,d, e^{\alpha-1}\,] = 1, b^{-1}db = e, b^{-1}eb = e^{\alpha}d\ \rangle.$

Adding another new generator $y = e^{\alpha-1}$ we get :

$$N \cong \langle\ b, d, e, y\ \mid\ b^n = (eb)^n = [\,d, y\,] = 1, e^{\alpha-1} = y, b^{-1}db = e, b^{-1}eb = yed\ \rangle.$$

Now, $e = cb^{-1} = abab^{-1} = [\,a^{-1}, b^{-1}\,]$, since $a^2 = 1$. Therefore, $e \in G'$.

Similarly, $d = beb^{-1} = b[\,a^{-1}, b^{-1}\,]b^{-1} \in G'$.

From (1) it is easily seen that $G/G' \cong \langle\ a, b\ \mid\ a^2 = b^n = [\,a, b\,] = 1\ \rangle$. Thus, $|G/G'| = 2n$. Now, $N/\langle\ d, e\ \rangle = \langle\ b\ \mid\ b^n = 1\ \rangle$, so $|N/\langle\ d, e\ \rangle| = n$. Since $|G/N| = 2$ and $|G/G'| = 2n$, $|N/G'| = n$, so it follows that $\langle\ d, e\ \rangle = G'$.

$[\,y, d\,] = 1$ in N, and $[\,y, e\,] = 1$ since $y = e^{\alpha-1}$, therefore $\langle\ y\ \rangle$ is central in $G' = \langle\ d, e\ \rangle$.

It has been shown that $|G':G''|$ is finite (Proposition 3.2 of [7]), so $y^i \in G''$ for some $i \geq 1$. Then $y^i \in G'' \cap Z(G')$, so G' is a stem extension of $G'/\langle\ y^i\ \rangle$ and hence G is finite if and only if $G'/\langle\ y^i\ \rangle$ is finite. $\langle\ y^i\ \rangle \leq \langle\ y\ \rangle$ and $|\langle\ y\ \rangle : \langle\ y^i\ \rangle|$ is obviously finite, so G' is finite if and only if $G'/\langle\ y\ \rangle$ is finite. Define $Y = \langle\ y\ \rangle^N$, then Y is the smallest normal subgroup in N containing $\langle\ y\ \rangle$. It can be shown from the relations in N that any word of the form $b^{\varepsilon_1}e^{\varepsilon_2}$ or $b^{\varepsilon_3}d^{\varepsilon_4}$ can be rewritten in the form $w_5(y, e, d)b^{\varepsilon_6}$ or $w_7(y, e, d)b^{\varepsilon_8}$ respectively, where

$w_5(y,e,d)$ and $w_7(y,e,d)$ are words in the generators y, e and d alone. Thus, any element of Y can be rewritten in the form :

$$b^{-i}\, w(y,e,d)^{-1}\, y^j\, w(y,e,d)\, b^i\,.$$

Since y commutes with e and d, this is the same as :

$$b^{-i}\, [w(y,e,d)^{-1}\, w(y,e,d)\,]\, y^j\, b^i = b^{-i}\, y^j\, b^i\,.$$

Now, $(b^{-i}\, y\, b^i)^j = b^{-i}\, y^j\, b^i$ for all values of j, so Y is finitely generated by

$$\langle\ y\,,\ b^{-1}yb,\ \ldots\ ,\ b^{-(n-1)}yb^{n-1}\ \rangle.$$

Let $n^{-1}yn \in Y$, $n \in N$. Now, $ngn^{-1} \in G'$, $\forall\ g \in G'$, since G' is normal in N and $\langle\ y\ \rangle$ is central in G', so $[\,y\,,\,ngn^{-1}\,] = 1$.

$$\begin{aligned}
&\therefore\ (n^{-1}yn)^{-1}\, g\, (n^{-1}yn)\\
&=\ n^{-1}y^{-1}(ngn^{-1})yn\\
&=\ n^{-1}y^{-1}y(ngn^{-1})n\\
&=\ g.
\end{aligned}$$

Therefore, $n^{-1}yn \in Z(G')$, so Y is normal in Z(G'), and hence Y is abelian.

Thus G' is finite if and only if G'/Y is finite, and hence N is finite if and only if N/Y is finite.

$$\bar{N} = N/Y = \langle\ \beta,\ \delta,\ \varepsilon\ \mid\ \beta^n = (\varepsilon\beta)^n = \varepsilon^{\alpha-1} = 1,\ \beta^{-1}\delta\beta = \varepsilon,\ \beta^{-1}\varepsilon\beta = \varepsilon\delta\ \rangle$$

Using $\beta^{-1}\delta\beta = \varepsilon$, $(\varepsilon\beta)^n = 1 \Rightarrow (\delta\beta)^n = 1 \Rightarrow (\beta^{-1}\delta^{-1})^n = 1$. Now we introduce $\gamma = \delta^{-1}$ and $\eta = \varepsilon^{-1}$ and eliminate $\delta = \gamma^{-1}$, $\varepsilon = \eta^{-1}$ to obtain :

$$\bar{N} = \langle\ \beta,\ \gamma,\ \eta\ \mid\ \beta^n = (\beta^{-1}\gamma)^n = \eta^{\alpha-1} = 1,\ \beta^{-1}\gamma\beta = \eta,\ \beta^{-1}\eta\beta = \gamma\eta\ \rangle.$$

Now, $\eta^{\alpha-1} = (\beta\gamma\beta^{-1})^{\alpha-1}$ using the fourth relation in the above presentation

$$\begin{aligned}
&\Rightarrow\quad \beta\gamma^{\alpha-1}\beta^{-1} = 1\\
&\Rightarrow\quad \gamma^{\alpha-1} = 1.
\end{aligned}$$

Thus $\eta^{\alpha-1} = 1$ can be replaced by $\gamma^{\alpha-1} = 1$ in the presentation. Eliminating $\eta = \beta^{-1}\gamma\beta$ we get :

$$\bar{N} = \langle\ \beta,\ \gamma\ \mid\ \beta^n = (\beta^{-1}\gamma)^n = \gamma^{\alpha-1} = 1,\ \beta^{-2}\gamma\beta^2 = \gamma\beta^{-1}\gamma\beta\ \rangle.$$

Now we introduce one final generator $\tau = \gamma^{-1}\beta$ and eliminate $\gamma = \beta\tau^{-1}$ to obtain :

$$\bar{N} = \langle\ \beta,\ \tau\ \mid\ \beta^n = \tau^n = (\beta\tau^{-1})^{\alpha-1} = 1,\ \tau\beta^2 = \beta\tau^2\ \rangle.$$

Now G is finite if and only if N is finite, and N is finite if and only if $\bar{N}$ is finite, thus $G(\alpha, n)$ is finite if and only if $\bar{N}(\alpha, n)$ is finite.

I determined the structure of the groups $\bar{N}(\alpha, n)$ for a variety of small values of α and n.

§5.2 The Groups $\bar{N}$ in General

At this point I would like to change the notation from that used in [7]. In what follows I will write the generators β and τ as a and b respectively and use β to denote α-1. Thus,

$$\bar{N}(\beta, n) \cong \bar{N}(\alpha-1, n) \cong \langle\ a, b\ \mid\ a^n = b^n = (ab^{-1})^\beta = 1,\ ab^2 = ba^2\ \rangle. \qquad (2)$$

Now,

$$\bar{N} / \bar{N}' = \langle\ a, b\ \mid\ a^n = b^n = (ab^{-1})^\beta = 1,\ ab^2 = ba^2\ ,\ ab = ba\ \rangle.$$

From the last two relations, $b = a$, so

$$\bar{N}/\bar{N}' = \langle\ a\ \mid\ a^n = 1\ \rangle,$$

$$\text{i.e.}\quad \bar{N} / \bar{N}' \cong C_n\ .$$

Therefore, carrying out a Todd-Coxeter coset enumeration on $\bar{N}$, enumerating over the subgroup $\bar{N}'$, we obtain a coset table isomorphic to the following :

	a	b
1	2	2
2	3	3
⋮	⋮	⋮
n-1	n	n
n	1	1

Now, using the Reidemeister-Schreier process, assuming that the original definitions were all made in the 'a' column, we can write in the Schreier generators as follows :

	a	b
1	2	$x_1.2$
2	3	$x_2.3$
⋮	⋮	⋮
n-1	n	$x_{n-1}.n$
n	y.1	$x_n.1$

Now we trace through the group relators for $\bar{N}$ to find relations for the subgroup $\bar{N}'$.

From $a^n = 1$ we obtain the relation $y = 1$, so y is trivial.

From $b^n = 1$ we obtain the relation $\prod_{i=1}^{n} x_i = 1$.

From $(ab^{-1})^\beta = 1$ we get $x_i^\beta = 1,\ 1 \le i \le n$.

Then from the last relation, $ab^2 = ba^2$, we obtain the series of relations $x_i = x_{i+1}\,x_{i+2}$, $1 \le i \le n$, where the subscripts are reduced modulo n.

Therefore,

$$\bar{N}'(\beta, n) = \langle\, x_i,\ 1 \le i \le n \mid \prod_{i=1}^{n} x_i = 1,\ x_i^\beta = 1,\ x_i = x_{i+1}\,x_{i+2} \,\rangle. \qquad (3)$$

Theorem 2

If n is odd then $\bar{N}(\beta, n)$ is finite for all n and is metabelian.

Proof

Looking at the last set of relations in (3), $x_i = x_{i+1}\,x_{i+2}$, $1 \le i \le n$:

$$\begin{aligned} x_{n-1} &= x_n x_1 \\ &= (x_1 x_2) x_1 \end{aligned}$$

$$
\begin{aligned}
&= x_1 x_2 (x_2 x_3) \\
&= x_1 x_2 (x_3 x_4) x_3 \\
&= x_1 x_2 x_3 x_4 (x_4 x_5) \\
&= \ldots\ldots \\
&= \begin{cases} x_1 x_2 x_3 x_4 \ldots x_{i-2} x_{i-1} x_i x_{i-1} & \text{i even} \\ x_1 x_2 x_3 x_4 \ldots x_{i-2} x_{i-1}^2 x_i & \text{i odd} \end{cases} \qquad (4)
\end{aligned}
$$

Therefore, if n is odd,

$$
\begin{aligned}
x_{n-1} &= x_1 x_2 x_3 x_4 \ldots x_{n-2} x_{n-1}^2 x_n \\
&= (x_1 x_2 x_3 x_4 \ldots x_{n-2} x_{n-1}) x_{n-1} x_n \\
&= x_n^{-1} x_{n-1} x_n \qquad \text{from the first relation in (3)} \\
\Rightarrow \quad x_n x_{n-1} &= x_{n-1} x_n \\
&\text{i.e. } [\, x_n , x_{n-1} \,] = 1.
\end{aligned}
$$

$\bar{N}'(\beta, n)$ can always be rewritten as a presentation on any two generators, so without loss of generality it can be rewritten in terms of x_n and x_{n-1}, so is abelian. Therefore $\bar{N}(\beta, n)$ is metabelian.

Since $\bar{N}'(\beta, n)$ is abelian, any product of the generators x_i, $1 \le i \le n$, can be reduced to the form :

$$x_1^{\alpha_1} x_2^{\alpha_2} \ldots\ldots x_{n-1}^{\alpha_{n-1}} x_n^{\alpha_n} \qquad \text{where } 0 \le \alpha_i < \beta \,,\ 1 \le i \le n$$

and as $\bar{N}'(\beta, n)$ can always be rewritten as a presentation on any two generators, this can be further simplified to :

$$x_i^{\gamma_i} x_j^{\gamma_j} \qquad \text{where } 0 \le i < \beta \,,\ 0 \le j < \beta .$$

Thus there are a finite number of elements in $\bar{N}'(\beta, n) \Rightarrow \bar{N}(\beta, n)$ is finite. ◊

Corollary

From (4), if n is even,

$$
\begin{aligned}
&x_{n-1} = x_1 x_2 x_3 x_4 \ldots x_{n-2} x_{n-1} x_n x_{n-1} \\
\Rightarrow \quad &\prod_{i=1}^{n} x_i = 1, \quad \text{i.e. the first relation in (3) is redundant.}
\end{aligned}
$$

§5.3 The Structure of $\bar{N}(\beta, n)$ for Small values of β

I will now look at the structure of $\bar{N}(\beta, n)$ for $\beta = 1, 2, 3$, and 4. The latter two are especially interesting.

$\beta = 1$

Here, $\bar{N}'(1, n) = \langle x_i, 1 \le i \le n \mid \prod_{i=1}^{n} x_i = 1, x_i = 1, x_i = x_{i+1} x_{i+2} \rangle \cong 1.$

Thus, $\bar{N}(1, n) \cong C_n .$

$\beta = 2$

Here, $\bar{N}'(2, n) = \langle x_i, 1 \le i \le n \mid \prod_{i=1}^{n} x_i = 1, x_i^2 = 1, x_i = x_{i+1} x_{i+2} \rangle.$ (5)

$x_i = x_{i+1} x_{i+2}$

$\Rightarrow \quad x_i^2 = x_i x_{i+1} x_{i+2}$

$\Rightarrow \quad x_i x_{i+1} x_{i+2} = 1$ since $x_i^2 = 1$. (6)

$x_i = x_{i+1} x_{i+2}$

$\Rightarrow \quad x_i x_{i+3} = x_{i+1} x_{i+2} x_{i+3} = 1$ since $x_i x_{i+1} x_{i+2} = 1$ from (6)

$\Rightarrow \quad x_i = x_{i+3}$ since $x_{i+3}^2 = 1$.

So $x_i = x_{i+3}$ for all i. Since $x_i = x_{i+n}$ for all i, $x_i = x_{i+t}$ where $t = (n, 3)$. Hence there are two distinct cases to investigate.

(i) $n \equiv 0 \pmod 3$

Here $t = 3$, so in this case the presentation (5) reduces to :

$$\langle x_1, x_2, x_3 \mid x_1^2 = x_2^2 = x_3^2 = 1, x_1 = x_2 x_3, x_2 = x_3 x_1, x_3 = x_1 x_2 \rangle.$$

The relation $\prod_{i=1}^{n} x_i = 1$ disappears since it is equivalent to $(x_1 x_2 x_3)^{\frac{n}{3}} = 1$, and we already know that $x_1 x_2 x_3 = 1$ from (6).

Eliminating $x_3 = x_1 x_2$ we obtain

$$\bar{N}'(2, n) = \langle x_1, x_2 \mid x_1^2 = x_2^2 = (x_1 x_2)^2 = 1 \rangle \cong C_2 \times C_2 .$$

(ii) $n \equiv 1 \pmod 3$ or $n \equiv 2 \pmod 3$

Here $t = (n, 3) = 1$, so $x_i = x_{i+1}$, $1 \le i \le n$. Therefore, $\bar{N}'(2, n)$ is trivial.

Hence, $\bar{N}'(2, n) \cong C_2 \times C_2$ if $n \equiv 0 \pmod 3$, otherwise it is trivial.

$\beta = 3$

Here, $$\bar{N}'(3,n) = \langle x_i, 1 \le i \le n \mid \prod_{i=1}^{n} x_i = 1, x_i^3 = 1, x_i = x_{i+1} x_{i+2} \rangle. \qquad (7)$$

I managed to show that $x_i = x_{i+8}$ $(1 \le i \le n)$ in $\bar{N}'(3,n)$. This means that we need only consider eight different cases, namely the eight possible values of n (mod 8). Before proving this I will first deduce two other relations which hold in $\bar{N}'(3,n)$ for all values of n.

(a) $(x_i x_{i+1})^3 = 1$ $(1 \le i \le n)$ (b) $(x_i x_{i+1}^{-1})^3 = 1$ $(1 \le i \le n)$

To prove (a), we use the facts that $x_i x_{i+1} = x_{i-1}$ and that $x_{i-1}^3 = 1$.

To prove (b), $x_i x_{i+1}^{-1} = x_i x_{i+1}^{-1} x_i^{-1} x_i = x_i x_{i-1}^{-1} x_i = x_i x_{i-1}^{-1} (x_{i-1} x_i)^{-1} x_{i-1} x_i^{-1}$.

$\therefore (x_i x_{i+1}^{-1})^3 = (x_i x_{i-1}^{-1} (x_{i-1} x_i)^{-1} x_{i-1} x_i^{-1})^3 = x_i x_{i-1}^{-1} (x_{i-1} x_i)^{-3} x_{i-1} x_i^{-1} = 1$ by (a).

Now, we have the following :

$$\begin{aligned}
x_n &= x_1 x_2 \\
x_{n-1} &= x_n x_1 \\
&= x_1 x_2 x_1 \\
x_{n-2} &= x_{n-1} x_n \\
&= (x_1 x_2 x_1)(x_1 x_2) \\
&= x_1 x_2 x_1^{-1} x_2 && \text{since } x_1^3 = 1 \\
x_{n-3} &= x_{n-2} x_{n-1} \\
&= (x_1 x_2 x_1^{-1} x_2)(x_1 x_2 x_1) \\
&= x_1 x_2 x_1^{-1} (x_2 x_1 x_2 x_1) \\
&= x_1 x_2 x_1^{-1} x_1^{-1} x_2^{-1} && \text{since } (x_2 x_1)^3 = 1 \text{ by (a)} \\
&= x_1 x_2 x_1 x_2^{-1} && \text{since } x_1^3 = 1
\end{aligned}$$

$= (x_1 x_2 x_1 x_2) x_2$ since $x_2^3 = 1$

$= x_2^{-1} x_1^{-1} x_2$ since $(x_1 x_2)^3 = 1$ from (a)

$x_{n-4} = x_{n-3} x_{n-2}$

$= (x_2^{-1} x_1^{-1} x_2)(x_1 x_2 x_1^{-1} x_2)$

$= x_2^{-1} x_1^{-1} (x_1^{-1} x_2^{-1} x_1^{-1}) x_1^{-1} x_2$ since $(x_2 x_1)^3 = 1$ from (a)

$= x_2^{-1} x_1 x_2^{-1} x_1 x_2$ since $x_1^3 = 1$

$= x_1^{-1} x_2 x_2$ since $(x_2^{-1} x_1)^3 = 1$ from (b)

$= x_1^{-1} x_2^{-1}$ since $x_2^3 = 1$

$x_{n-5} = x_{n-4} x_{n-3}$

$= (x_1^{-1} x_2^{-1})(x_2^{-1} x_1^{-1} x_2)$

$= x_1^{-1} x_2 x_1^{-1} x_2$ since $x_2^3 = 1$

$= x_2^{-1} x_1$ since $(x_1^{-1} x_2)^3 = 1$ from (b)

$x_{n-6} = x_{n-5} x_{n-4}$

$= (x_2^{-1} x_1)(x_1^{-1} x_2^{-1})$

$= x_2$ since $x_2^3 = 1$

$x_{n-7} = x_{n-6} x_{n-5}$

$= x_2 x_2^{-1} x_1$

$= x_1$

So $x_i = x_{i+8}$ for all i. But we also have that $x_i = x_{i+n}$ for all i, so that $x_i = x_{i+t}$ for all i, where $t = (n, 8)$. If n is odd, $t = 1$, and we immediately deduce that $x_i = x_j$ for any i and j, and that $\bar{N}'(3, n)$ is trivial. Thus, we need only look at even values of n.

(i) $\underline{n \equiv 0 \pmod 8}$

Since n is even, $\prod_{i=1}^{n} x_i = 1$ is redundant. Therefore the presentation for $\bar{N}'(3, n)$ is

$$\langle x_i, 1 \le i \le 8 \mid x_i^3 = 1, x_i = x_{i+1} x_{i+2} \rangle.$$

Eliminating generator $x_8 = x_1 x_2$ and $x_3 = x_2^{-1} x_1$ we have the following presentation :

$$\langle x_1, x_2, x_4, x_5, x_6, x_7 \mid x_1^3 = x_2^3 = (x_1 x_2^{-1})^3 = x_4^3 = x_5^3 = x_6^3 = x_7^3 = (x_1 x_2)^3 = 1,$$
$$x_2 = x_2^{-1} x_1 x_4,\ x_2^{-1} x_1 = x_4 x_5,\ x_4 = x_5 x_6,\ x_5 = x_6 x_7,\ x_6 = x_7 x_1 x_2,\ x_7 = x_1 x_2 x_1 \rangle.$$

We now delete the generators $x_7 = x_1 x_2 x_1$ and $x_4 = x_1^{-1} x_2^2$ to get :

$\langle x_1, x_2, x_5, x_6 \mid x_1^3 = x_2^3 = (x_1 x_2^{-1})^3 = (x_1^{-1} x_2^2)^3 = x_5^3 = x_6^3 = (x_1 x_2 x_1)^3 = (x_1 x_2)^3 = 1, x_2^{-1} x_1 = x_1^{-1} x_2^2 x_5, x_1^{-1} x_2^2 = x_5 x_6, x_5 = x_6 x_1 x_2 x_1, x_6 = x_1 x_2 x_1^2 x_2 \rangle$.

Now, since $x_2^3 = 1$, the fourth and the eighth relations are the same. From the first and the third relations we deduce that $(x_1 x_2^{-1})^3 = (x_1^{-2} x_2^{-1})^3 = 1 \Rightarrow (x_2 x_1^2)^3 = 1$ i.e. the seventh relation is also redundant. Thus, rewriting x_1^2 as x_1^{-1} and x_2^2 as x_2^{-1}, we have :

$$\langle x_1, x_2, x_5, x_6 \mid x_1^3 = x_2^3 = (x_1 x_2^{-1})^3 = x_5^3 = x_6^3 = (x_1 x_2)^3 = 1, \\ x_2^{-1} x_1 = x_1^{-1} x_2^{-1} x_5, x_1^{-1} x_2^{-1} = x_5 x_6, x_5 = x_6 x_1 x_2 x_1, x_6 = x_1 x_2 x_1^{-1} x_2 \rangle.$$

We can now eliminate x_5 and x_6 using the seventh and the tenth relations respectively.

From $x_5 = x_6 x_1 x_2 x_1$ we then obtain :

$$\begin{aligned} & x_2 x_1 x_2^{-1} x_1 = x_1 x_2 x_1^{-1} x_2 x_1 x_2 x_1 \\ \Rightarrow \quad & (x_2 x_1 x_2) x_1^{-1} x_2^{-1} x_1 x_2^{-1} x_1^{-1} = 1 && \text{since } x_1^3 = x_2^3 = 1 \\ \Rightarrow \quad & (x_1^{-1} x_2^{-1} x_1^{-1}) x_1^{-1} x_2^{-1} x_1 x_2^{-1} x_1^{-1} = 1 && \text{since } (x_2 x_1)^3 = 1 \\ \Rightarrow \quad & (x_1 x_2^{-1})^3 = 1 && \text{since } x_1^3 = 1. \end{aligned}$$

This relation is redundant since it is the same as the third relation.

From $x_1^{-1} x_2^{-1} = x_5 x_6$ we get

$$\begin{aligned} & x_1^{-1} x_2^{-1} = x_2 x_1 x_2^{-1} x_1 x_1 x_2 x_1^{-1} x_2 \\ \Rightarrow \quad & (x_1^{-1} x_2^{-1} x_1^{-1}) x_2 x_1 x_2^{-1} x_1 x_2 = 1 && \text{since } x_2^3 = x_1^3 = 1 \\ \Rightarrow \quad & x_2 x_1 x_2 x_2 x_1 x_2^{-1} x_1 x_2 = 1 && \text{since } (x_2 x_1)^3 = 1 \\ \Rightarrow \quad & (x_1 x_2^{-1})^3 = 1 && \text{since } x_2^3 = 1. \end{aligned}$$

This is the same as the third relation, so is also redundant.

$$\begin{aligned} x_5^3 = 1 \quad & \Rightarrow (x_2 x_1 x_2^{-1} x_1)^3 = 1 \\ & \Rightarrow ((x_1 x_2 x_1 x_2) x_2)^3 = 1 && \text{since } x_2^3 = 1 \\ & \Rightarrow (x_2^{-1} x_1^{-1} x_2)^3 = 1 && \text{since } (x_1 x_2)^3 = 1 \\ & \Rightarrow (x_1^{-1})^3 = 1 && \text{which is the same as the first relator .} \end{aligned}$$

Thus, $x_5^3 = 1$ is also redundant.

We show that $x_6^3 = 1$ is redundant in exactly the same way :

$$\begin{aligned} x_6^3 = 1 \quad & \Rightarrow (x_1 x_2 x_1^{-1} x_2)^3 = 1 \\ & \Rightarrow ((x_2 x_1 x_2 x_1) x_1)^3 = 1 && \text{since } x_1^3 = 1 \\ & \Rightarrow (x_1^{-1} x_2^{-1} x_1)^3 = 1 && \text{since } (x_1 x_2)^3 = 1 \\ & \Rightarrow (x_2^{-1})^3 = 1 && \text{which is the same as the second relator.} \end{aligned}$$

Thus, our presentation becomes :

$$\langle x_1, x_2 \mid x_1^3 = x_2^3 = (x_1 x_2^{-1})^3 = (x_1 x_2)^3 = 1 \rangle$$

which is a metabelian group of order 27.

$\bar{N}''(3,n) \cong C_3$ and $\bar{N}' / \bar{N}'' \cong C_3 \times C_3$.

(ii) $n \equiv 2$ or $6 \pmod 8$

In both cases $t=2$, so $x_i = x_{i+2}$, $1 \le i \le n$. Since $x_i = x_{i+1} x_{i+2}$, $x_{i+2} = x_{i+1} x_{i+2} \Rightarrow$ $x_{i+1} = 1$ for all i. Therefore, $\bar{N}'(3,n)$ is trivial.

(iii) $n \equiv 4 \pmod 8$

Here $t=4$, so $x_i = x_{i+4}$, $1 \le i \le n$. Thus, we have the following presentation :

$$\langle x_1, x_2, x_3, x_4, \mid x_1^3 = x_2^3 = x_3^3 = x_4^3 = 1, x_1 = x_2 x_3, x_2 = x_3 x_4, x_3 = x_4 x_1, x_4 = x_1 x_2 \rangle. \quad \ldots\ldots\ldots\ldots\ldots (8)$$

$x_2 = x_3 x_4 \Rightarrow x_2 = x_3 x_1 x_2$ since $x_4 = x_1 x_2$

$\Rightarrow x_3 = x_1^{-1}$.

$x_1 = x_2 x_3 \Rightarrow x_1 = x_2 x_1^{-1}$ since $x_3 = x_1^{-1}$

$\Rightarrow x_2 = x_1^2 = x_1^{-1}$ since $x_1^3 = 1$.

Thus, $x_4 = x_1 x_2 \Rightarrow x_4 = x_1 x_1^{-1} = 1$ so x_4 is trivial.

From the last four relations in (8) we then get $x_1 = x_2 = x_3$, but since we know that $x_3 = x_1^{-1}$, $x_1 = x_2 = x_3 = 1$. Thus, $\bar{N}'(3,n)$ is trivial.

Hence, $\bar{N}'(3,n)$ is trivial unless $n \equiv 0 \pmod 8$, in which case it is a metabelian group of order 27.

$\beta = 4$

$$\bar{N}'(4, n) = \langle x_i, 1 \le i \le n \mid \prod_{i=1}^{n} x_i = 1, x_i^4 = 1, x_i = x_{i+1} x_{i+2} \rangle$$

Here I have managed to show that $x_i = x_{i+12}$, $1 \le i \le n$, where the subscripts are reduced modulo n. Again I will first deduce some short relations which are useful in the proof.

(a) $(x_i x_{i+1})^4 = 1$ (b) $(x_{i+1} x_i^2)^4 = 1$ (c) $(x_i x_{i+1}^{-1})^4 = 1$ (d) $(x_{i+1}^2 x_i)^4 = 1$

(a) This can be proved immediately from $(x_{i-1})^4 = 1$ and $x_{i-1} = x_i x_{i+1}$.

(b) From (a) we have $(x_{i-1} x_i)^4 = 1$. Substituting for $x_{i-1} = x_i x_{i+1}$, the result follows.

(c) $x_{i+2}^4 = 1$. Using the relation $x_i = x_{i+1} x_{i+2}$ to substitute for x_{i+2} the result follows.

(d) From (c), $(x_{i+1} x_{i+2}^{-1})^4 = 1$, and again we substitute for x_{i+2} using $x_i = x_{i+1} x_{i+2}$. This gives $(x_{i+1}^2 x_1^{-1})^4 = 1$, so taking the inverse, and using the fact that $x_{i+1}^4 = 1$, we get the required result.

$$x_n = x_1 x_2$$

$$\begin{aligned} x_{n-1} &= x_n x_1 \\ &= x_1 x_2 x_1 \end{aligned}$$

$$\begin{aligned} x_{n-2} &= x_{n-1} x_n \\ &= (x_1 x_2 x_1)(x_1 x_2) \\ &= x_1 x_2 x_1^2 x_2 \end{aligned}$$

$$\begin{aligned} x_{n-3} &= x_{n-2} x_{n-1} \\ &= (x_1 x_2 x_1^2 x_2)(x_1 x_2 x_1) \\ &= x_1 x_2 x_1^2 x_2 x_1 x_2 x_1 \end{aligned}$$

$$\begin{aligned} x_{n-4} &= x_{n-3} x_{n-2} \\ &= (x_1 x_2 x_1^2 x_2 x_1 x_2 x_1)(x_1 x_2 x_1^2 x_2) \\ &= x_1 x_2 x_1^2 x_2 x_1^{-1} (x_1^2 x_2 x_1^2 x_2 x_1^2 x_2) \\ &= x_1 x_2 x_1^2 x_2 x_1^{-1} x_2^{-1} x_1^2 && \text{since } (x_1^2 x_2)^4 = 1 \text{ from (b) and } x_1^4 = 1 \\ &= x_1 (x_2 x_1^2)^2 x_1 x_2^{-1} x_1^2 && \text{since } x_1^4 = 1 \\ &= x_1 (x_1^2 x_2^{-1})^2 x_1 x_2^{-1} x_1^2 && \text{since } (x_2 x_1^2)^4 = 1 \text{ from (b) and } x_1^4 = 1 \\ &= x_1^{-1} x_2^{-1} x_1 (x_1 x_2^{-1})^2 x_1^2 && \text{since } x_1^4 = 1 \\ &= x_1^{-1} x_2^{-1} x_1 x_2 x_1^{-1} x_2 x_1 && \text{since } (x_1 x_2^{-1})^4 = 1 \text{ from (c)} \end{aligned}$$

$$\begin{aligned} x_{n-5} &= x_{n-4} x_{n-3} \\ &= (x_1^{-1} x_2^{-1} x_1 x_2 x_1^{-1} x_2 x_1)(x_1 x_2 x_1^2 x_2 x_1 x_2 x_1) \\ &= x_1^{-1} x_2^{-1} x_1 x_2 x_1 (x_1^2 x_2)^3 x_1 x_2 x_1 && \text{since } x_1^4 = 1 \\ &= x_1^{-1} x_2^{-1} x_1 x_2 x_1 (x_2^{-1} x_1^{-2}) x_1 x_2 x_1 && \text{since } (x_2 x_1^2)^4 = 1 \text{ from (b)} \end{aligned}$$

$$= x_1^{-1}x_2^{-1}x_1x_2x_1x_2^{-1}x_1^{-1}x_2x_1$$

$$\begin{aligned} x_{n-6} &= x_{n-5}x_{n-4} \\ &= (x_1^{-1}x_2^{-1}x_1x_2x_1x_2^{-1}x_1^{-1}x_2x_1)(x_1^{-1}x_2^{-1}x_1x_2x_1^{-1}x_2x_1) \\ &= x_1^{-1}x_2^{-1}x_1x_2^2x_1 \end{aligned}$$

$$\begin{aligned} x_{n-7} &= x_{n-6}x_{n-5} \\ &= (x_1^{-1}x_2^{-1}x_1x_2^2x_1)(x_1^{-1}x_2^{-1}x_1x_2x_1x_2^{-1}x_1^{-1}x_2x_1) \\ &= x_1^{-1}x_2^{-1}x_1x_2x_1x_2x_1x_2^{-1}x_1^{-1}x_2x_1 \\ &= x_1^{-1}x_2^2(x_2x_1)^3x_2^{-1}x_1^{-1}x_2x_1 && \text{since } x_2^4 = 1 \\ &= x_1^{-1}x_2^2x_1^{-1}x_2^{-1}x_2^{-1}x_1^{-1}x_2x_1 && \text{since } (x_2x_1)^4 = 1 \text{ from (a)} \\ &= (x_1^{-1}x_2^2)^3x_2^{-1}x_1 && \text{since } x_2^4 = 1 \\ &= x_2^2x_1x_2^{-1}x_1 && \text{since } (x_1^{-1}x_2^2)^4 = 1 \text{ from (d) and } x_2^4 = 1 \end{aligned}$$

$$\begin{aligned} x_{n-8} &= x_{n-7}x_{n-6} \\ &= (x_2^2x_1x_2^{-1}x_1)(x_1^{-1}x_2^{-1}x_1x_2^2x_1) \\ &= x_2^2x_1x_2^2x_1x_2^2x_1 && \text{since } x_2^4 = 1 \\ &= x_1^{-1}x_2^2 && \text{since } (x_2^2x_1)^4 = 1 \text{ from (d) and } x_2^4 = 1 \end{aligned}$$

$$\begin{aligned} x_{n-9} &= x_{n-8}x_{n-7} \\ &= (x_1^{-1}x_2^2)(x_2^2x_1x_2^{-1}x_1) \\ &= x_2^{-1}x_1 && \text{since } x_2^4 = 1 \end{aligned}$$

$$\begin{aligned} x_{n-10} &= x_{n-9}x_{n-8} \\ &= (x_2^{-1}x_1)(x_1^{-1}x_2^2) \\ &= x_2 \end{aligned}$$

$$\begin{aligned} x_{n-11} &= x_{n-10}x_{n-9} \\ &= x_2(x_2^{-1}x_1) \\ &= x_1 \end{aligned}$$

Thus, in $\bar{N}'(4,n)$, $x_i = x_{i+12} \pmod n$ for all i, so we will only have to consider the twelve cases corresponding to the possible values of n (mod 12). In fact, since $x_i = x_{i+n}$, we have $x_i = x_{i+t}$, where $t = (12,n)$, so we shall only need to consider the cases $t = 0, 1, 2, 3, 4$ and 6.

(i) $n \equiv 0 \pmod{12}$

$$\bar{N}'(4,n) \cong \bar{N}'(4,12) = \langle x_i, 1 \le i \le 12 \mid x_i^4 = 1, x_i = x_{i+1} x_{i+2} \rangle$$

We can use the computer programs described in Chapters 2 and 3 to show that this group is infinite. This is done most easily using the Reidemeister-Schreier method.

First of all the presentation is simplified using the Tietze transformation program, NTTRANS, to get the following :

$$\begin{aligned}\bar{N}' \cong \langle x_1, x_2 \mid\ & x_1^4 = x_2^4 = (x_1 x_2)^4 = (x_1 x_2^{-1})^4 = (x_1^2 x_2^{-1})^4 = (x_1 x_2^{-2})^4 = \\ & x_1^2 x_2 x_1^{-1} x_2^{-1} x_1 x_2^{-1} x_1^{-1} x_2 x_1^{-1} x_2 x_1 x_2 x_1 x_2^{-1} x_1^{-1} x_2^{-2} x_1^{-1} x_2 x_1^{-1} x_2^{-1} x_1 x_2 = \\ & (x_1 x_2 x_1^{-1} x_2 x_1 x_2^{-1})^4 = 1 \rangle. \end{aligned} \tag{9}$$

Then the commutator $[x_1, x_2]$ is added as a relator to give a presentation for $\bar{N}'/\bar{N}''$. This is not simplified at this stage.

The Todd-Coxeter program is used to find the order of $\bar{N}'/\bar{N}''$, which turns out to be 16.

We then find a presentation for $\bar{N}''$. This is done by carrying out the Reidemeister-Schreier option on the coset table, and tracing out the subgroup relations from the relations in (9). [To do this we remove the relator $[x_1, x_2] = 1$ from the presentation for $\bar{N}'/\bar{N}''$ which we have just enumerated, before we call Reidemeister-Schreier. This was the reason for not simplifying the presentation earlier.]

$$\begin{aligned}\bar{N}'' \cong \langle y_1, y_2, y_3, y_4, y_5 \mid\ & y_1^2 y_3^2 = y_1 y_4 y_1^{-1} y_4 = y_2^2 y_3^2 = y_4 y_3 y_4 y_3^{-1} = \\ & y_4 y_5 y_4^{-1} y_5^{-1} = y_1 y_3^{-1} y_2 y_1 y_3^{-1} y_2^{-1} = y_1 y_5^{-1} y_1 y_5^{-1} y_2^2 = y_2^3 y_4 y_2 y_4 = \\ & y_2^2 y_5 y_3^{-1} y_5 y_3^{-1} = y_2 y_3 y_5 y_3^{-1} y_2^{-1} y_5^{-1} = y_2 y_4 y_5^{-1} y_2^{-1} y_4 y_5^{-1} = \\ & y_1 y_2 y_3 y_1^{-1} y_4 y_2 y_4 y_3 = y_2^2 y_3^{-1} y_2^{-1} y_4 y_2 y_3^{-1} y_4^{-1} \rangle\end{aligned}$$

When we do simplify the presentation for $\bar{N}'/\bar{N}''$ using NTTRANS we obtain :

$$\langle x_1, x_2 \mid x_1^4 = x_2^4 = [x_1, x_2] = (x_1 x_2)^4 = (x_1 x_2^{-1})^4 = (x_1 x_1 x_2^{-1})^4 = (x_1 x_2^{-1} x_2^{-1})^4 = 1 \rangle.$$

It is obvious that the last four relations are redundant, so we have :

$$\bar{N}'/\bar{N}'' \cong \langle x_1, x_2 \mid x_1^4 = x_2^4 = [x_1, x_2] = 1 \rangle \cong C_4 \times C_4.$$

This process of adding commutator relations and carrying out the Reidemeister-Schreier process is repeated to find a presentation for $\bar{N}'''$. We find that

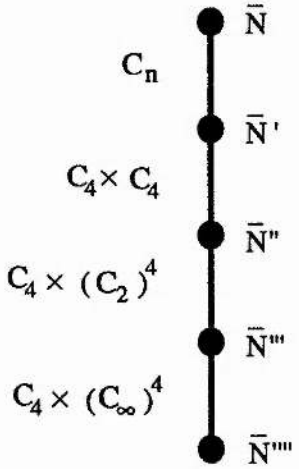
$$\bar{N}''/\bar{N}''' \cong C_4 \times (C_2)^4.$$

Then, $\bar{N}'''/\bar{N}''''$ is calculated in the same way to get :

$$\langle z_1, z_2, z_3, z_4, z_5 \mid z_1^4 = [z_1, z_2] = [z_1, z_3] = [z_1, z_4] = [z_1, z_5] = [z_2, z_3] = [z_2, z_4] = [z_2, z_5] = [z_3, z_4] = [z_3, z_5] = [z_4, z_5] = 1 \rangle.$$

$$\cong C_4 \times C_\infty \times C_\infty \times C_\infty \times C_\infty.$$

Thus, $\bar{N}'(4,n)$ is infinite when $n \equiv 0 \pmod{12}$.



(ii) <u>$n \equiv 1 \pmod{12}$, $n \equiv 5 \pmod{12}$, $n \equiv 7 \pmod{12}$, $n \equiv 11 \pmod{12}$</u>

Here $t = (n, 12) = 1$, so $x_i = x_{i+1}$ for all i. Therefore, $\bar{N}'(4,n)$ is trivial in each case.

(iii) <u>$n \equiv 2 \pmod{12}$, $n \equiv 10 \pmod{12}$</u>

In these two cases, $t = (n, 12) = 2$. Therefore, $x_i = x_{i+2}$ for all i. The relation $\prod_{i=1}^{n} x_i = 1$ is redundant since n is even, so we get the presentation :

$$\langle x_1, x_2 \mid x_1^4 = x_2^4 = 1, x_1 = x_2 x_1, x_2 = x_1 x_2 \rangle.$$

The last two relations imply that $x_1 = x_2 = 1$. Thus, $\bar{N}'(4,n)$ is trivial when $n \equiv 2 \pmod{12}$ or $n \equiv 10 \pmod{12}$.

(iv) <u>$n \equiv 3 \pmod{12}$, $n \equiv 9 \pmod{12}$</u>

Here $n = 3m$, where m is odd, so $t = (12, 3m) = 3$. Therefore, $x_i = x_{i+3}$ for all i, so

the relation $\prod_{i=1}^{n} x_i = 1$ gives us $(x_1 x_2 x_3)^m = 1$.

$\therefore \quad \bar{N}'(4,n) \cong \langle x_1, x_2, x_3 \mid (x_1 x_2 x_3)^m = 1,\ x_1^4 = x_2^4 = x_3^4 = 1,\ x_1 = x_2 x_3,$

$x_2 = x_3 x_1,\ x_3 = x_1 x_2 \rangle.$

Substituting $x_1 = x_2 x_3$ into the first relation we get $(x_1)^{2m} = 1$. Since m is odd and $x_1^4 = 1$ we have $x_1^2 = 1$.

Eliminating $x_3 = x_1 x_2$ we get :

$$\langle x_1, x_2 \mid x_1^2 = x_2^2 = [x_1, x_2] = 1 \rangle \cong C_2 \times C_2.$$

Therefore, $\bar{N}'(4,n) \cong C_2 \times C_2$ if $n \equiv 3 \pmod{12}$ or $n \equiv 9 \pmod{12}$.

(v) <u>$n \equiv 4 \pmod{12}$, $n \equiv 8 \pmod{12}$</u>

Here $t = (n, 12) = 4$, so $x_i = x_{i+4}$ for all i. Since n is even $\prod_{i=1}^{n} x_i = 1$ is redundant, so we have the presentation :

$$\langle x_1, x_2, x_3, x_4 \mid x_1^4 = x_2^4 = x_3^4 = x_4^4 = 1,\ x_1 = x_2 x_3,\ x_2 = x_3 x_4,\ x_3 = x_4 x_1,\ x_4 = x_1 x_2 \rangle.$$

Here, $x_4 = x_1 x_2$ so $x_2 = x_3 x_4 = x_3 x_1 x_2 \Rightarrow x_3 = x_1^{-1}$.

Now, $x_1 = x_2 x_3 \Rightarrow x_1 = x_2 x_1^{-1} \Rightarrow x_2 = x_1^2$. Thus $x_4 = x_1 x_2 = x_1^3$ so x_2, x_3 and x_4 are all redundant. From the seventh relator $x_3 = x_4 x_1$ we get $x_1^{-1} = x_1^3 x_1 \Rightarrow x_1^5 = 1$. Thus, since $x_1^4 = 1$, $x_1 = 1$ and so $\bar{N}'(4,n)$ is trivial in this case.

(vi) <u>$n \equiv 6 \pmod{12}$</u>

Here $t = (n, 12) = 6$, so $x_i = x_{i+6}$. Since n is even $\prod_{i=1}^{n} x_i = 1$ is redundant, so we have the presentation :

$$\langle x_1, x_2, x_3, x_4, x_5, x_6 \mid x_1^4 = x_2^4 = x_3^4 = x_4^4 = x_5^4 = x_6^4 = 1,\ x_1 = x_2 x_3,\ x_2 = x_3 x_4,\ x_3 = x_4 x_5,\ x_4 = x_5 x_6,\ x_5 = x_6 x_1,\ x_6 = x_1 x_2 \rangle.$$

Eliminate $x_6 = x_1 x_2$ to get :

$$\langle x_1, x_2, x_3, x_4, x_5 \mid x_1^4 = x_2^4 = x_3^4 = x_4^4 = x_5^4 = (x_1 x_2)^4 = 1,\ x_1 = x_2 x_3,\ x_2 = x_3 x_4,\ x_3 = x_4 x_5,\ x_4 = x_5 x_1 x_2,\ x_5 = x_1 x_2 x_1 \rangle.$$

Eliminate $x_5 = x_1 x_2 x_1$ to get :

$$\langle x_1, x_2, x_3, x_4 \mid x_1^4 = x_2^4 = x_3^4 = x_4^4 = (x_1^2 x_2)^4 = (x_1 x_2)^4 = 1, x_1 = x_2 x_3, x_2 = x_3 x_4, x_3 = x_4 x_1 x_2 x_1, x_4 = x_1 x_2 x_1^2 x_2 \rangle.$$

Eliminate $x_4 = x_1 x_2 x_1^2 x_2$ to get :

$$\langle x_1, x_2, x_3 \mid x_1^4 = x_2^4 = x_3^4 = (x_1 x_2 x_1^2 x_2)^4 = (x_1^2 x_2)^4 = (x_1 x_2)^4 = 1, x_1 = x_2 x_3, x_2 = x_3 x_1 x_2 x_1^2 x_2, x_3 = x_1 x_2 x_1^2 x_2 x_1 x_2 x_1 \rangle.$$

Eliminate $x_3 = x_2^{-1} x_1$ to get :

$$\langle x_1, x_2 \mid x_1^4 = x_2^4 = (x_2^{-1} x_1)^4 = (x_1 x_2 x_1^2 x_2)^4 = (x_1^2 x_2)^4 = (x_1 x_2)^4 = 1, x_2^{-1} x_1^2 x_2 x_1^2 = 1, x_2^{-1} x_1 = x_1 x_2 x_1^2 x_2 x_1 x_2 x_1 \rangle.$$

From the seventh relation in this presentation, using the fact that $x_1^4 = 1$ we get :

$$x_1^2 x_2 = x_2 x_1^2.$$

The last relation in the presentation can then be simplified as follows :

$$\begin{aligned}
& x_2^{-1} x_1 = x_1 x_2 x_1^2 x_2 x_1 x_2 x_1 \\
\Rightarrow \quad & x_1 x_2 (x_1^2 x_2) x_1 x_2^2 = 1 \\
\Rightarrow \quad & x_1 x_2 (x_2 x_1^2) x_1 x_2^2 = 1 && \text{since } x_1^2 x_2 = x_2 x_1^2 \\
\Rightarrow \quad & x_1 x_2^2 x_1^{-1} x_2^2 = 1 && \text{since } x_1^4 = 1 \\
\Rightarrow \quad & x_1 x_2^2 = x_2^2 x_1 && \text{since } x_2^4 = 1.
\end{aligned}$$

We now show that the third, fourth and fifth relations are redundant.

$$\begin{aligned}
& (x_2^{-1} x_1)^4 = 1 \\
\Rightarrow \quad & (x_2^3 x_1 x_2^{-1} x_1)^2 = 1 && \text{since } x_2^4 = 1 \\
\Rightarrow \quad & (x_2 (x_1 x_2^2) x_2^{-1} x_1)^2 = 1 && \text{since } x_2^2 x_1 = x_1 x_2^2 \\
\Rightarrow \quad & (x_2 x_1)^4 = 1
\end{aligned}$$

This is just the sixth relation, so is redundant.

$$\begin{aligned}
& (x_1 x_2 x_1^2 x_2)^4 = 1 \\
\Rightarrow \quad & (x_1^3 x_2^2)^4 = 1 && \text{since } x_1^2 x_2 = x_2 x_1^2 \\
\Rightarrow \quad & (x_1^3 x_2^2 x_1^3 x_2^2)^2 = 1 \\
\Rightarrow \quad & (x_1^6 x_2^4)^2 = 1 && \text{since } x_2^2 x_1 = x_1 x_2^2 \\
\Rightarrow \quad & x_1^{12} = 1 && \text{since } x_2^4 = 1
\end{aligned}$$

Therefore, since $x_1^4 = 1$, this relation is redundant.

$$
\begin{aligned}
& (x_1^2 x_2)^4 = 1 \\
\Rightarrow \quad & (x_1^2 x_2 x_1^2 x_2)^2 = 1 \\
\Rightarrow \quad & (x_1^4 x_2^2)^2 = 1 && \text{since } x_2 x_1^2 = x_1^2 x_2 \\
\Rightarrow \quad & x_2^4 = 1 && \text{since } x_1^4 = 1
\end{aligned}
$$

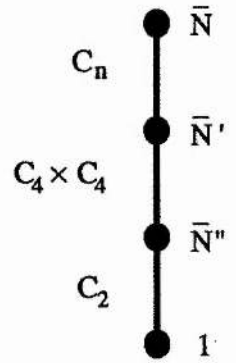
This is the same as the second relation, so is redundant.

Thus we have the presentation :

$$\langle x_1, x_2 \mid x_1^4 = x_2^4 = (x_1 x_2)^4 = 1,\ x_2 x_1^2 = x_1^2 x_2,\ x_1 x_2^2 = x_2^2 x_1 \rangle.$$

Using Todd-Coxeter process, $|\bar{N}'(4,n)| = 32$ and from the Reidemeister-Schreier method,

$$\bar{N}' / \bar{N}'' \cong \langle x_1, x_2 \mid x_1^4 = x_2^4 = [x_1, x_2] = 1 \rangle \cong C_4 \times C_4 \quad \text{and} \quad \bar{N}'' \cong C_2.$$



Hence $\bar{N}'(4,n)$ is trivial unless $n \equiv 0 \pmod 3$.

Unfortunately, values of $\beta \geq 5$ do not seem to give this type of cyclic presentation. In the next section we will look at certain examples with $\beta \geq 5$.

§5.4 The Structure of $\bar{N}(\beta, n)$ for Small Values of n

When n is odd, we see from the proof of Theorem 2 that the relation $\prod_{i=1}^{n} x_i = 1$ can be reduced using the relations $x_i = x_{i+1}\, x_{i+2}$, $1 \le i \le n$, to any of the relations $[x_i, x_{i+1}] = 1$. Now,

$$\begin{aligned} x_n &= x_1 x_2 \\ x_{n-1} = x_n x_1 = x_1 x_2 x_1 &= x_1^2 x_2 \quad \text{since } [x_1, x_2] = 1 \\ x_{n-2} = x_{n-1} x_n = x_1^2 x_2 x_1 x_2 &= x_1^3 x_2^2 \\ x_{n-3} = x_{n-2} x_{n-1} = x_1^3 x_2^2 x_1^2 x_2 &= x_1^5 x_2^3 \\ &= \ldots\ldots . \end{aligned}$$

From this it can be seen that the exponents of x_1 and x_2 are successive elements of the Fibonacci sequence defined as :

$$f_1 = 1, \qquad f_2 = 1, \qquad f_{n+2} = f_n + f_{n+1} .$$

The first few terms of this sequence are

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, \ldots .$$

Thus, it can be seen that $x_{n-i} = x_1^{f_{i+2}} x_2^{f_{i+1}}$.

Therefore, when we eliminate all of the generators, except x_1 and x_2, from the presentation, the relations $x_i = x_{i+1}\, x_{i+2}$ reduce to :

$$x_1 = x_1^{f_{n+1}} x_2^{f_n} \qquad \text{and} \qquad x_2 = x_1^{f_n} x_2^{f_{n-1}} .$$

Thus, our presentation for $\bar{N}'(\beta, n)$ reduces to :

$$\bar{N}'(\beta, n) = \langle x_1, x_2 \mid x_1^{\beta} = x_2^{\beta} = 1,\ x_1 = x_1^{f_{n+1}} x_2^{f_n},\ x_2 = x_1^{f_n} x_2^{f_{n-1}},\ [x_1, x_2] = 1 \rangle . \quad (10)$$

We will look for the moment at the group defined by the presentation :

$$\langle x_1, x_2 \mid x_1 = x_1^{f_{n+1}} x_2^{f_n},\ x_2 = x_1^{f_n} x_2^{f_{n-1}},\ [x_1, x_2] = 1 \rangle .$$

The order of this group is given by the absolute value of the determinant of the relation matrix:

$$\begin{vmatrix} f_{n+1} - 1 & f_n \\ f_n & f_{n-1} - 1 \end{vmatrix}$$

$$= |(f_{n+1}-1)(f_{n-1}-1)-f_n^2|$$
$$= |f_{n+1}f_{n-1}-f_n^2-(f_{n-1}+f_{n+1})+1|$$
$$= |(-1)^n-(f_{n-1}+f_{n+1})+1| \text{ since } f_{n+1}f_{n-1}-f_n^2=(-1)^n$$
$$= f_{n-1}+f_{n+1} \qquad \text{when n is odd}$$
$$= g_n$$

where g_n is the nth Lucas number given by the formula :

$$g_1 = 1, \qquad g_2 = 3, \qquad g_{n+2} = g_n + g_{n+1}\,.$$

The first few terms of this sequence are :

1, 3, 4, 7, 11, 18, 29, 47, 89, 76, 123, 199, 322, 521, 843 ... .

Hence, if n is odd, the order of $\bar{N}'(\beta, n)$ divides g_n and depends on the value of β. If g_n is coprime to β, obviously $\bar{N}'(\beta, n)$ is trivial. Thus, if g_n is prime, $\bar{N}'(\beta, n)$ is trivial, except for those values of β for which $(g_n, \beta) = g_n$, in which case it is the cyclic group of order g_n.

n=1

From (2) $a^n = b^n = 1 \Rightarrow a = b = 1$, so the group $\bar{N}(\beta, 1)$ is trivial for all values of β.

n=2

$$\bar{N}(\beta,2) = \langle a, b \mid a^2 = b^2 = (ab^{-1})^\beta = 1,\ ab^2 = ba^2 \rangle$$

Using the first two relations the last relation reduces to a=b, therefore $\bar{N}(\beta, 2) \cong C_2$.

n=3

Here $g_n = 4$, so $\bar{N}'(\beta, 3)$ is trivial unless $(\beta, 4) \neq 1$. If $(\beta, 4) = 2$ or 4 we get the following presentation for $\bar{N}'(\beta, 3)$ from (10) :

$$\bar{N}'(\beta,3) = \langle x_1, x_2 \mid x_1^{2m} = x_2^{2m} = 1,\ x_1 = x_1^3 x_2^2,\ x_2 = x_1^2 x_2,\ [x_1, x_2] = 1 \rangle.$$

From the fourth relation, $x_1^2 = 1$, and substituting this into the third relation we get $x_2^2 = 1$. Thus, the first two relations are redundant and we have :

$$\bar{N}'(\beta,3) = \langle x_1, x_2 \mid x_1^2 = x_2^2 = [x_1, x_2] = 1 \rangle \cong C_2 \times C_2.$$

n=4

$$\bar{N}'(\beta,4) = \langle x_1, x_2, x_3, x_4 \mid x_1^\beta = x_2^\beta = x_3^\beta = x_4^\beta = 1,\ x_1 = x_2 x_3,\ x_2 = x_3 x_4,$$
$$x_3 = x_4 x_1,\ x_4 = x_1 x_2 \rangle$$

Since n is even, $\prod_{i=1}^{4} x_i = 1$ is redundant.

When $\beta = 4$ this is the same presentation as that obtained for $\bar{N}'(4,n)$ when $n \equiv 4, 8 \pmod{12}$. Looking back at the working for this it is clear that $\bar{N}'(\beta,4)$ is trivial unless $\beta \equiv 0 \pmod 5$ in which case $\bar{N}'(\beta,4) \cong C_5$.

n=5

Here $g_n = 11$ so $\bar{N}'(\beta,5)$ is trivial unless β is a multiple of 11, in which case $\bar{N}'(\beta,5) \cong C_{11}$.

n=6

$$\bar{N}'(\beta,6) = \langle x_i,\ 1 \le i \le 6 \mid \prod_{i=1}^{6} x_i = 1,\ x_i^\beta = 1,\ x_i = x_{i+1} x_{i+2} \rangle$$
$$= \langle x_1, x_2, x_3, x_4, x_5, x_6 \mid x_1^\beta = x_2^\beta = x_3^\beta = x_4^\beta = x_5^\beta = x_6^\beta = 1,$$
$$x_1 = x_2 x_3,\ x_2 = x_3 x_4,\ x_3 = x_4 x_5,\ x_4 = x_5 x_6,\ x_5 = x_6 x_1,\ x_6 = x_1 x_2 \rangle$$

Again, $\prod_{i=1}^{6} x_i = 1$ is redundant since n is even.

Eliminating the generators x_6, x_5, x_4 and x_3 using the same substitutions as in the working for $\bar{N}'(4,n)$ when $n \equiv 6 \pmod{12}$ we get the following presentation :

$$\langle x_1, x_2 \mid x_1^\beta = x_2^\beta = (x_2^{-1} x_1)^\beta = (x_1 x_2 x_1^2 x_2)^\beta = (x_1^2 x_2)^\beta = (x_1 x_2)^\beta = 1,$$
$$x_2^{-1} x_1^2 x_2 x_1^2 = 1,\ x_2^{-1} x_1 = x_1 x_2 x_1^2 x_2 x_1 x_2 x_1 \rangle.$$

Using the seventh relation, the last relation can be simplified :

$$x_2^{-1} x_1 = x_1 x_2 x_1^2 x_2 x_1 x_2 x_1$$
$$\Rightarrow\ x_1 x_2 x_1^2 x_2 x_1 x_2^2 = 1$$
$$\Rightarrow\ x_1 x_2^2 (x_2^{-1} x_1^2 x_2 x_1^2) x_1^{-1} x_2^2 = 1$$
$$\Rightarrow\ x_1 x_2^2 x_1^{-1} x_2^2 = 1 \qquad \text{since } x_2^{-1} x_1^2 x_2 x_1^2 = 1.$$

Now, $(x_1 x_2 x_1^2 x_2)^\beta = 1$

$$\Rightarrow\ (x_1 x_2^2 x_1^{-2})^\beta = 1 \qquad \text{since } x_2^{-1} x_1^2 x_2 x_1^2 = 1$$

$$\Rightarrow (x_1^{-1}x_2^{-2})^{\beta}=1$$
$$\Rightarrow (x_2^2x_1)^{\beta}=1.$$

Thus we now have :

$$\bar{N}'(\beta, 6) \cong \langle x_1, x_2 \mid x_1^{\beta}=x_2^{\beta}=(x_2^{-1}x_1)^{\beta}=(x_2^2x_1)^{\beta}=(x_1^2x_2)^{\beta}=(x_1x_2)^{\beta}=1,$$
$$x_2^{-1}x_1^2x_2x_1^2=1,\ x_1x_2^2x_1^{-1}x_2^2=1\rangle. \qquad (11)$$

There are two cases to consider here, namely when β is odd and when β is even.

(i) β odd

Here $\beta=2m+1$ where $m\geq 0$.

Then, $(x_2^2x_1)^{\beta}=1$
$$\Rightarrow x_2^2x_1(x_2^2x_1x_2^2x_1)^m=1$$
$$\Rightarrow x_2^2x_1(x_1^2)^m=1 \qquad \text{since } x_1x_2^2x_1^{-1}x_2^2=1$$
$$\Rightarrow x_2^2x_1^{\beta}=1$$
$$\Rightarrow x_2^2=1 \qquad \text{since } x_1^{\beta}=1.$$

Similarly, $(x_1^2x_2)^{\beta}=1 \Rightarrow x_1^2=1$.

Now, since β is odd and $x_1^{\beta}=x_2^{\beta}=1$, $x_1=x_2=1$. Hence, if β is odd, $\bar{N}'(\beta, 6)$ is trivial.

(ii) β even

In this case, $\bar{N}'/\bar{N}'' \cong \langle x_1, x_2 \mid x_1^4=x_2^4=x_1^{\beta}=x_2^{\beta}=[x_1, x_2]=1\rangle$.

If $\beta=2m$, m odd, $x_1^{\beta}=1$
$$\Rightarrow x_1^{2m}=1$$
$$\Rightarrow x_1^2(x_1^4)^{(m-1)/2}=1$$
$$\Rightarrow x_1^2=1 \qquad \text{since } x_1^4=1.$$

By symmetry, $x_2^2=1$.

$$\therefore\ \bar{N}'/\bar{N}'' \cong \langle x_1, x_2 \mid x_1^2=x_2^2=[x_1, x_2]=1\rangle \cong C_2\times C_2.$$

We now carry out the Reidemeister-Schreier process using a suitable coset table for $C_2\times C_2$ to find a presentation for $\bar{N}'$. Taking the original coset definitions to be $2=1x_1$, $3=1x_2$ and $4=3x_1$ we can define Schreier generators y_1, y_2, y_3, y_4 and y_5 as follows :

	x_1	x_2
1	2	3
2	$y_1.1$	$y_3.4$
3	4	$y_4.1$
4	$y_2.3$	$y_5.2$

We then obtain the following subgroup relations from (11) :

From 1st relation : $y_1^m = 1$, $y_2^m = 1$

2nd relation : $y_4^m = 1$, $(y_3 y_5)^m = 1$

3rd relation : $(y_3 y_2 y_4)^m = 1$, $(y_1 y_5)^m = 1$

4th relation : $(y_5^{-1} y_2)^m = 1$, $(y_1 y_4^{-1} y_3^{-1})^m = 1$

5th relation : $(y_1 y_2 y_4)^m = 1$, $(y_1 y_3 y_2 y_5)^m = 1$

6th relation : $(y_3 y_5 y_1 y_4)^m = 1$, $(y_5 y_3 y_2 y_4)^m = 1$

7th relation : $y_3 y_5 y_4 = 1$, $y_1 y_4 y_1^{-1} y_3 y_5 = 1$, $y_5 y_3 y_4 = 1$, $y_2 y_4 y_2^{-1} y_5 y_3 = 1$

8th relation : $y_1 y_2 = 1$, $y_1 y_3 y_2 y_3^{-1} = 1$, $y_2 y_4 y_1 y_4^{-1} = 1$, $y_2 y_5 y_1 y_5^{-1} = 1$

Now, $y_2 = y_1^{-1}$ from the first relation in the line above, which then gives us $[y_1, y_3] = [y_1, y_4] = [y_1, y_5] = 1$ from the other relations in that row.

From the second last set of relations, $y_3 y_5 y_4 = 1$ and $y_5 y_3 y_4 = 1$. From these $[y_3, y_4] = 1$, $[y_3, y_5] = 1$ and $[y_4, y_5] = 1$ can be easily deduced.

Therefore all of the generators commute. Using this fact and eliminating $y_2 = y_1^{-1}$ we obtain the following presentation :

$$\langle y_1, y_3, y_4, y_5 \mid y_1^m = y_3^m = y_4^m = y_5^m = 1, y_5 = y_4^{-1} y_3^{-1}, [y_1, y_3] = [y_1, y_4] = [y_1, y_5] = [y_3, y_4] = [y_3, y_5] = [y_4, y_5] = 1 \rangle.$$

Now we can eliminate $y_5 = y_4^{-1} y_3^{-1}$ to get :

$$\bar{N}''(\beta, 6) = \langle y_1, y_3, y_4 \mid y_1^m = y_3^m = y_4^m = [y_1, y_3] = [y_1, y_4] = [y_3, y_4] = 1 \rangle$$

$$\cong C_m \times C_m \times C_m .$$

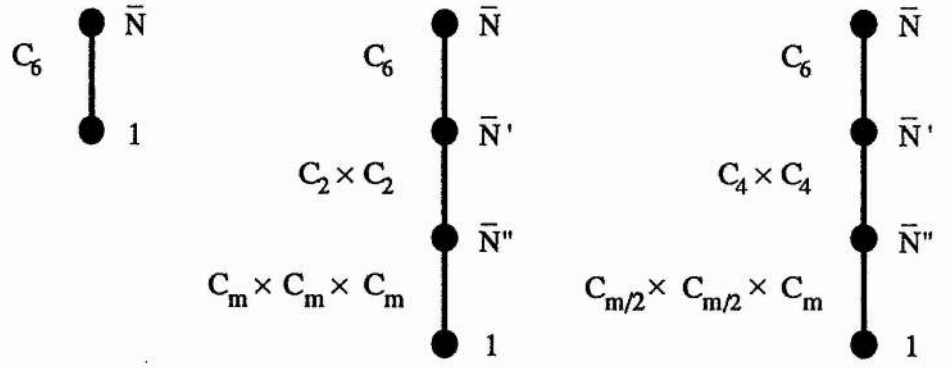
If $\beta = 2m$, where m is even, β is a multiple of 4 so the relations $x_1^{\beta} = 1$ and $x_2^{\beta} = 1$ are redundant.

$$\therefore\ \bar{N}' / \bar{N}'' \cong \langle x_1, x_2 \mid x_1^4 = x_2^4 = [x_1, x_2] = 1 \rangle \cong C_4 \times C_4 .$$

By computer methods the presentation for $\bar{N}''(\beta, 6)$ was found to be :

$\langle y_1, y_2, y_3 \mid y_1^{m/2} = (y_2 y_3)^{m/2} = (y_1 y_2 y_3^{-1})^{m/2} = [y_1, y_2] = [y_1, y_3] = [y_2, y_3] = 1 \rangle$.

$\cong C_{m/2} \times C_{m/2} \times C_m$.



β odd $\qquad$ $\beta = 2m$, m odd $\qquad$ $\beta = 2m$, m even

n=7

In this case $g_n = 29$, so $\bar{N}'(\beta, 7)$ is trivial unless β is a multiple of 29, whereupon $\bar{N}'(\beta, 7) \cong C_{29}$.

n=8

Here we have the presentation :

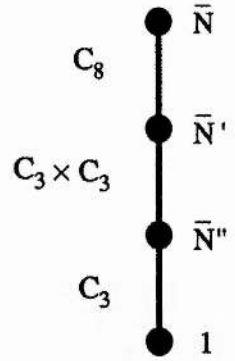
$$\bar{N}'(\beta, 8) = \langle x_i,\ 1 \le i \le 8 \mid \prod_{i=1}^{8} x_i = 1,\ x_i^{\beta} = 1,\ x_i = x_{i+1} x_{i+2} \rangle .$$

From Section 5.3 we can state immediately what this group is for $1 \le \beta \le 4$.

$\beta = 1$: $\bar{N}'(1, 8)$ is trivial.

$\beta = 2$: $n \equiv 2 \pmod 3$ so $\bar{N}'(2, 8)$ is trivial.

$\beta = 3$: $n \equiv 0 \pmod 8$ so $\bar{N}'(3, 8)$ has order 27.



$\underline{\beta = 4}$: $n \equiv 8 \pmod{12}$ so $\bar{N}'(4, 8)$ is trivial.

$\underline{\beta = 5}$: By computer methods $\bar{N}'(5, 8)$ is infinite. I showed this using the modified Todd-Coxeter algorithm. First of all I entered the presentation for $\bar{N}'(5,8)$ into NTTRANS and eliminated x_i, $3 \leq i \leq 8$. Then I added the relation $[x_1, x_2] = 1$ to this and used MODTC to find the order of $\bar{N}'/\bar{N}''$. This turned out to be 5, so $\bar{N}'/\bar{N}'' \cong C_5$. The coset table obtained was :

	x_1	x_2
1	5	4
2	1	5
3	2	1
4	3	2
5	4	3

I then removed the commutator relator from the presentation and added the following subgroup generators :

$$y_1 = x_2^{-1} x_1^2, \quad y_2 = x_2^{-2} x_1^4, \quad y_3 = x_2 x_1^3, \quad y_4 = x_2^{-3} x_1 .$$

It can be easily seen that these give the same coset definitions as those in the above table. [Remember that all defining is done on the backward scan of a subgroup generator or relator.] I changed from the ordinary Todd-Coxeter algorithm to the modified algorithm at this point and obtained a presentation for $\bar{N}''(5,8)$ on these generators (and some high numbered generators of course). This presentation was reduced to one on y_1, y_2, y_3 and y_4 alone using NTTRANS and the following relations were obtained :

$y_1^2 y_2^{-1} y_4 y_2^{-1} y_4^{-1}$ $\quad$ $y_1^2 y_4^{-1} y_2^{-1} y_4 y_2^{-1}$

$y_1 y_2^{-1} y_1 y_3 y_2 y_3^{-1}$ $\quad$ $y_1 y_2^{-1} y_1 y_3^{-1} y_2 y_3$

$(y_1 y_2^{-1} y_4)^2$ $\quad$ $(y_1 y_3 y_2^{-1})^2$

$y_1 y_3 y_4^{-1} y_3 y_1^{-1} y_4$ $\quad$ $y_1 y_3^{-1} y_1 y_4^{-2} y_3$

$(y_1 y_4^{-1} y_3)^2$ $\quad$ $(y_2 y_3 y_4^{-1})^2$

$y_2 y_4 y_2^{-1} y_4 y_3^{-2}$ $\quad$ $y_1 y_3^2 y_1^{-1} y_4^{-1} y_3^2 y_4$

$(y_1 y_2^{-1} y_3)^2$

There were no coincidences found during the enumeration so there was no point in using the extra relation facility. I then repeated this process to find a presentation for $\bar{N}'''(5, 8)$. This time the coset table for $\bar{N}''/\bar{N}'''$ is a bit larger, so it is more difficult to choose suitable subgroup generators.

	y_1	y_2	y_3	y_4
1	5	4	6	2
2	12	3	14	1
3	9	2	15	4
4	8	1	7	3
5	1	8	11	12
6	11	7	1	14
8	4	5	10	9
9	3	12	16	8
10	7	11	8	16
11	6	10	5	13
12	2	9	13	5
13	14	16	12	11
14	13	15	2	6
15	16	14	3	7
16	15	13	9	10

From this table it can be seen that each generator is an involution, so

$$\bar{N}''/\bar{N}''' \cong C_2 \times C_2 \times C_2 \times C_2 .$$

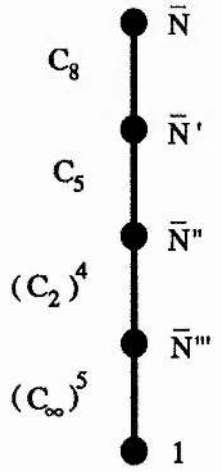
The subgroup generators I added were :

$$z_1 = y_4^2,\ z_2 = y_2 y_4 y_2 y_4,\ z_3 = y_1^2,\ z_4 = y_3 y_2^2 y_3,\ z_5 = y_4 y_2 y_1 y_4 y_1 y_2,\ z_6 = y_2^2 .$$

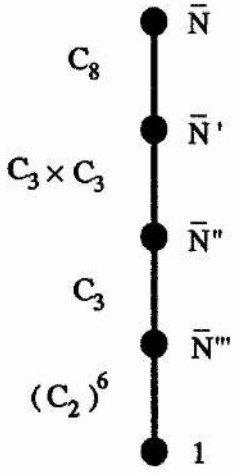
When we simplify the presentation obtained from the modified algorithm we can eliminate generator z_4, which is redundant, and carry out some substring searching to obtain the following for $\bar{N}'''(5,8)$:

$$\langle z_1, z_2, z_3, z_5, z_6 \mid [z_1, z_2] = [z_1, z_3] = [z_1, z_5] = [z_1, z_6] = [z_2, z_3] = [z_2, z_5] = [z_2, z_6] = [z_3, z_5] = [z_3, z_6] = [z_5, z_6] = 1 \rangle \cong (C_\infty)^5 .$$

Thus, $\bar{N}'''(5,8)$ is infinite, implying that $\bar{N}(5,8)$ is infinite. R.M. Thomas managed to prove that this group was infinite by hand. His proof takes up 10 pages in [7] !



$\underline{\beta = 6}$: By computer methods $\bar{N}'(6, 8)$ has order $2^6 \times 3^3$.



$\underline{\beta = 7}$: $\bar{N}'(7, 8)$ is trivial.

$\underline{\beta = 8}$: $\bar{N}'(8, 8)$ is trivial.

$\underline{\beta = 9}$: From the computer, $\bar{N}'(9, 8)$ appears to be infinite.

$\underline{\beta = 10}$: From the computer, $\bar{N}'(10, 8)$ appears to be infinite.

n=9

In this case $g_n = 76$, so the order of $\bar{N}'(\beta, 9)$ must divide 76. From (10) we have :

$$\bar{N}'(\beta, 9) = \langle x_1, x_2 \mid x_1^{\beta} = x_2^{\beta} = 1, x_1 = x_1^{55} x_2^{34}, x_2 = x_1^{34} x_2, [x_1, x_2] = 1 \rangle.$$

With the help of the computer the following results were obtained :

If $(\beta, 76) = 1$ $\bar{N}'(\beta, 9)$ is trivial.

If $(\beta, 38) = 2$ $\bar{N}'(\beta, 9) \cong C_2 \times C_2$.

If $(\beta, 38) = 19$ $\bar{N}'(\beta, 9) \cong C_{19}$.

If $(\beta, 38) = 38$ $\bar{N}'(\beta, 9) \cong C_{38} \times C_2 \cong C_{19} \times C_2 \times C_2$.

n=10

Here we have the presentation :

$$\bar{N}'(\beta, 10) = \langle x_i, 1 \le i \le 10 \mid \prod_{i=1}^{10} x_i = 1, x_i^{\beta} = 1, x_i = x_{i+1} x_{i+2} \rangle.$$

From Section 5.3 it can be seen that $\bar{N}'(\beta, 10)$ is trivial when β=1, 2, 3 or 4.

$\underline{\beta = 5}$: From the computer, $\bar{N}'(5, 10)$ has order 62400. The presentation obtained from the machine is :

$$\langle x_1, x_2 \mid x_1^5 = x_2^5 = (x_1 x_2)^5 = (x_1 x_2^{-1})^5 = (x_1^2 x_2)^5 = (x_1 x_2^{-2})^5 =$$
$$(x_1^2 x_2 x_1 x_2)^5 = (x_1 x_2^{-1} x_1 x_2^{-2})^5 = (x_1 x_2^2 x_1 x_2^{-2})^5 =$$
$$(x_1 x_2 x_1^{-1} x_2^2 x_1^{-1} x_2 x_1 x_2^{-2})^5 = x_1^2 x_2 x_1 x_2^2 x_1^{-1} x_2 x_1 x_2^{-1} x_1 x_2^{-1} x_1 x_2 x_1^{-1} x_2^2 x_1 x_2 =$$
$$x_1^2 x_2 x_1^2 x_2^{-1} x_1 x_2^{-2} x_1 x_2 x_1^{-1} x_2^2 x_1^{-1} x_2 x_1 x_2^{-2} x_1 x_2^{-1} = 1 \rangle.$$

Two elements, y_1 and y_2, with orders 2 and 3 respectively, were found to generate the group.

$$y_1 = x_2^{-1} x_1^{-1} x_2^{-2} x_1 x_2^{-1} x_1^{-2} x_2^{-2} x_1 x_2^{-1} x_1^{-1} x_2 x_1^{-1} x_2^3 x_1^{-1}$$

$$y_2 = x_2 x_1^2 x_2$$

These satisfy the relations for PSU(3, 4) in presentation 12.2 in [16] :

$$\langle y_1, y_2 \mid y_1^2 = y_2^3 = (y_1 y_2)^{15} = (y_1^{-1} y_2^{-1} y_1 y_2)^5 = ((y_1 y_2)^3 (y_1 y_2^{-1})^3)^3 = (y_1 y_2^{-1} (y_1 y_2)^5)^4 = 1 \rangle.$$

so it is PSU(3, 4).

$\underline{\beta = 6}$: It is possible to find a subgroup K of $\bar{N}'$ such that the order of $\bar{N}'/K = 100$ and $\bar{K}/K' \cong C_\infty$ (by computer), so $\bar{N}'(6, 8)$ must be infinite.

n=11

In this case $g_n = 199$ so, since 199 is prime, $\bar{N}'(\beta, 11)$ is trivial unless β is a multiple of 199, whereupon $\bar{N}'(\beta, 11) \cong C_{199}$.

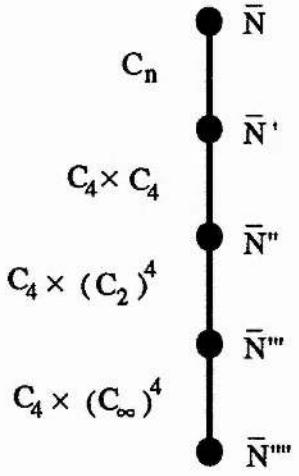
n=12

$\underline{\beta = 1}$: $\bar{N}'(1, 12)$ is trivial.

$\underline{\beta = 2}$: $n \equiv 0 \pmod 3$ so $\bar{N}'(2, 12) \cong C_2 \times C_2$.

$\underline{\beta = 3}$: $n \equiv 4 \pmod 8$ so $\bar{N}'(3, 12)$ is trivial.

$\underline{\beta = 4}$: From Section 5.3 $\bar{N}'(4, 12)$ is infinite and $\bar{N}(4, 12)$ has the following structure :



n=13

In this case $g_n = 521$ so, since 521 is prime, $\bar{N}'(\beta, 13)$ is trivial unless β is a multiple of 521, whereupon $\bar{N}'(\beta, 13) \cong C_{521}$.

n=14

From Section 5.3, $\bar{N}'(\beta, 14)$ is trivial when $\beta = 1, 2, 3$ or 4.

$\beta = 5$: $\bar{N}'(5, 14)$ is perfect, i.e. $\bar{N}''(5, 14) = \bar{N}'(5, 14)$.

n=15

In this case $g_n = 1364$, so the order of $\bar{N}'(\beta, 15)$ must divide $1364 = 2^2 \times 11 \times 31$. From (10) we have :

$$\bar{N}'(\beta, 15) = \langle x_1, x_2 \mid x_1^{\beta} = x_2^{\beta} = 1, x_1 = x_1^{987} x_2^{610}, x_2 = x_1^{610} x_2^{377}, [x_1, x_2] = 1 \rangle.$$

With the help of the computer the following results were obtained :

If $(\beta, 1364) = 1$ $\quad \bar{N}'(\beta, 15)$ is trivial.

If $(\beta, 682) = 2$ $\quad \bar{N}'(\beta, 15) \cong C_2 \times C_2$.

If $(\beta, 682) = 341$ $\quad \bar{N}'(\beta, 15) \cong C_{341} \cong C_{31} \times C_{11}$.

If $(\beta, 682) = 682$ $\quad \bar{N}'(\beta, 15) \cong C_{682} \cong C_{31} \times C_{11} \times C_2 \times C_2$.

If $(\beta, 62) = 31$ $\quad \bar{N}'(\beta, 15) \cong C_{31}$.

If $(\beta, 62) = 62$ $\quad \bar{N}'(\beta, 15) \cong C_{31} \times C_2 \times C_2$.

If $(\beta, 22) = 11$ $\quad \bar{N}'(\beta, 15) \cong C_{11}$.

If $(\beta, 22) = 22$ $\quad \bar{N}'(\beta, 15) \cong C_{11} \times C_2 \times C_2$.

Obviously this selection is just a small number of the results which can be obtained, either by hand or computer about these groups. However, as the values of β and n get larger, the task becomes harder.

References

The following references : [18], [23] and [27] are not referred to directly in the main text but were used for background information.

[1] D.G. Arrell, S. Manrai & M.F. Worboys, 'A procedure for obtaining simplified defining relations for a subgroup', in *Groups St Andrews 1981*, edited by C.M. Campbell & E.F. Robertson, L.M.S. Lecture Notes 71 (1982), Cambridge University Press, 155-159.

[2] D.G. Arrell & E.F. Robertson, 'A modified Todd-Coxeter algorithm', in *Computational Group Theory*, Academic Press, London (1984), 27-32.

[3] M.J. Beetham, 'Space saving in coset enumeration', in *Computational Group Theory*, Academic Press, London (1984), 19-25.

[4] M.J. Beetham & C.M. Campbell, 'A note on the Todd-Coxeter coset enumeration algorithm', *Proc. Edinburgh Math. Soc.* 20 (1976), 73-79.

[5] C.T. Benson & N.S. Mendelsohn, 'A calculus for a certain class of word problems in groups', *J. Combinatorial Theory* 1 (1966), 202-208.

[6] R. Brown, D.L. Johnson & E.F. Robertson, 'Some computations of non-abelian tensor products', *J. Algebra* 111 (1987), 177-202.

[7] C.M. Campbell, P.M. Heggie, E.F. Robertson & R.M. Thomas, '*One-relator products of cyclic groups and Fibonacci-like sequences*', (Technical Report 35 Dept. of Computing Studies, University of Leicester, April 1990).

[8] C.M. Campbell, P.M. Heggie, E.F. Robertson & R.M. Thomas, 'One-relator products of cyclic groups and Fibonacci-like sequences', in *Fibonacci Numbers* (D. Reidel), to appear.

[9] C.M. Campbell, P.M. Heggie, E.F. Robertson & R.M. Thomas, 'Finite one-relator products of two cyclic groups with the relator of arbitrary length', submitted.

[10] C.M. Campbell, P.M. Heggie, E.F. Robertson & R.M. Thomas, 'Cyclically presented groups embedded in one-relator products of cyclic groups', submitted.

[11] C.M. Campbell & E.F. Robertson, 'On the simple groups G with $|G| < 10^5$', *Groups - Korea 1983*, Springer Lect. Notes Math. 1098 (1984), 15-20.

[12] C.M. Campbell & E.F. Robertson, 'Presentations for the simple groups G, $10^5 < |G| < 10^6$', *Comm. Algebra* 12 (1984), 2643-2663.

[13] C.M. Campbell, E.F. Robertson & R.M. Thomas, *Group presentations and sequences of Fibonacci type* (Technical Report 8, Dept. of Computing Studies, University of Leicester, April 1988).

[14] C.M. Campbell, E.F. Robertson & R.M. Thomas, *The groups G(n;{1,-1}k, 2, -2) and G(n;{-1,1}k, 2, -2)*, (Technical Report 11, Dept. of Computing Studies, University of Leicester, August 1988).

[15] J.J. Cannon, L.A. Dimino, G. Havas & J.M. Watson, 'Implementation and analysis of the Todd-Coxeter algorithm', *Math. Comput.* 27 (1973), 463-490.

[16] J.J. Cannon, J. Mackay & K-C. Young, 'The non-abelian simple groups G, $|G| < 10^5$ - presentations', *Comm. Alg.* 7 (1979), 1397-1406.

[17] M.D.E. Conder, 'Three-relator quotients of the modular group', *Quart. J. Math.* 38 (1987), 427-447.

[18] H.S.M. Coxeter & W.O.J. Moser, *Generators and Relations for Discrete Groups*, 4th ed., Springer-Verlag, Berlin (1980).

[19] G. Havas, 'A Reidemeister-Schreier program', in *Proc. Second Internat. Conf. Theory of Groups*, edited by M.F. Newman, Lecture notes in Mathematics, Vol. 372, Springer-Verlag, Berlin (1974), 347-356.

[20] G. Havas, P.E. Kenne, J.S. Richardson & E.F. Robertson, 'A Tietze transformation program', *Computational Group Theory*, London: Academic Press (1984), 69-73.

[21] G.Havas, J.S.Richardson & L.S. Sterling, 'The last of the Fibonacci groups', *Proc. Roy. Soc. Edinburgh* 83A (1979) , 199-203.

[22] G. Havas & C.C. Sims, Communication regarding the bug in the Felsch strategy of the Todd-Coxeter program.

[23] D.L. Johnson, *Topics in the Theory of Group Presentations*, London Math. Soc. Lecture Note Series, Vol. 42, Cambridge University Press, Cambridge (1980).

[24] J. Leech, 'Coset enumeration on digital computers', *Proc. Cambridge Phil. Soc.* 59 (1963), 257-267.

[25] J. Leech, 'Computer proof of relations in groups', in *Topics in Group Theory and Computation*, edited by M.P.J. Curran, Academic Press, London (1977), 38-61.

[26] D.H. Maclain, 'A procedure for determining defining relations of a subgroup', *Glasgow Math. J.* 18 (1977), 51-56.

[27] W. Magnus, A. Karrass & D. Solitar, *Combinatorial Group Theory : Presentations of Groups in terms of Generators and Relations*, Pure and Appl. Math., Vol. 13, Interscience, New York (1966).

[28] N.S. Mendelsohn, 'An algorithmic solution for a word problem in group theory', *Canad. J. Math.* 16 (1964), 509-516. Correction, *Canad. J. Math.* 17 (1965), 505.

[29] N.S. Mendelsohn, 'Defining relations for subgroups of finite index of groups with a finite presentation', in *Computational Problems in Abstract Algebra*, edited by J. Leech, Pergamon, Oxford (1970), 43-44.

[30] J. Neubüser, 'An elementary introduction to coset table methods in computational Group Theory', in *Groups - St Andrews 1981*, edited by C.M. Campbell & E.F. Robertson, L.M.S. Lecture Notes Vol. 71 (1982), Cambridge University Press, 1-45.

[31] M.O. Rabin, 'Recursive unsolvability of group theoretic problems', *Ann. of Math.* 67 (1958), 172-194.

[32] E.F. Robertson, 'Tietze transformations with weighted substring search', *J. Symbolic Computation* 6 (1988), 59-64.

[33] S. Sidki, 'A generalization of the alternating groups - A question on finiteness and representation', *J. Algebra* 75 (1982), 324-372.

[34] J.A. Todd & H.S.M. Coxeter,' A practical method for enumerating cosets of a finite abstract group', *Proc. Edinburgh Math. Soc.* 5 (1936), 26-34.

[35] H.F. Trotter, 'A machine program for coset enumeration', *Canad. Math. Bull.* 7 (1964), 357-368.

[36] J.N. Ward, 'A note on the Todd-Coxeter algorithm', in *Group Theory*, edited by R.A. Bryce, J. Cossey and M.F. Newman, Lecture notes in Mathematics, Vol. 573, Springer-Verlag, Berlin (1977), 126-129.